TOULOUSE LAUTREC

APRENDIZAJE AUTOMATICO CON PYTHON

Ejercicio Gaussian Naive Bayes

Ing. Alexander Valdez

Curso 2290, Clases Lunes y Miercoles 20:00-22:30pm

Tercera Clase

Ejercicio Machine Learning: Gaussian Naive Bayes

El teorema de Bayes

- El teorema de Bayes es una ecuación que **describe la relación de probabilidades condicionales** de cantidades estadísticas. En clasificación bayesiana estamos interesados en encontrar la probabilidad de que ocurra una "clase" dadas unas características observadas (datos). Lo podemos escribir como **P(Clase | Datos)**. El teorema de Bayes nos dice cómo lo podemos expresar en términos de cantidades que podemos calcular directamente:

$$P(\text{Clase} \mid \text{Datos}) = \frac{P(\text{Datos}|\text{Clase}) * P(\text{Clase})}{P(\text{Datos})}$$

- **Clase** es una salida en particular, por ejemplo "comprar"
- **Datos** son nuestras características, en nuestro caso los ingresos, gastos, hijos, etc
- **P(Clase|Datos)** se llama posterior (y es el resultado que queremos hallar)
- **P(Datos|Clase)** se llama "verosimilitud" (en inglés likelihood)
- **P(Clase)** se llama anterior (pues es una probabilidad que ya tenemos)
- **P(Datos)** se llama probabilidad marginal

Posterior de comprar es lo que queremos hallar: P(comprar|datos).

Explicaremos los demá:

- **P(comprar)** es la probabilidad que ya tenemos. Es sencillamente el número de veces que se selecciona comprar =1 en nuestro conjunto de datos, dividido el total de observaciones. En nuestro caso (luego lo veremos en Python) son 67/202
- **p(ingresos|comprar)p(ahorros|comprar)p(hijos|comprar)** es la verosimilitud. Los nombres *Gaussian y Naive* (ingenuo) del algoritmo vienen de dos suposiciones:
  1. asumimos que las características de la verosimilitud **no estan correlacionada entre ellas**. Esto seria que los ingresos sean independientes a la cantidad de hijos y de los ahorros. Como no es siempre cierto y es una *suposición ingenua* es que aparece en el nombre "naive bayes"
  2. Asumimos que el valor de las características (ingresos, hijos, etc) tendrá una distribución normal (gaussiana). Esto nos permite calcular cada parte p(ingresos|comprar) usando la función de probabilidad de densidad normal.
- **probabilidad marginal** muchas veces es difícil de calcular, sin embargo, por la ecuación que vimos más arriba, no la necesitaremos para obtener nuestro valor a posterior. Esto simplifica los cálculos.

¿Comprar o alquilar casa? ¿Qué me conviene?

La explicacion completa en el blog www.aprendemachinelearning.com

En este ejercicio, usaremos el algoritmo de Gaussian Naive Bayes para decidir si nos conviene Alguilar o Comprar una casa

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
import seaborn as sb

%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import SelectKBest
```

Importemos Datos de entrada

```
from google.colab import drive
```

In [2]:
```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [6]:
```
dataframe = pd.read_csv(r"/content/drive/MyDrive/DATASET_TOULOUSE_C3/comprar_alquilar.cs
dataframe.head(10)
# REFERENCIAS:
# ingresos y gastos son mensuales de 1 personas o 2 si están casados.
# trabajo: 0-sin trabajo 1-autonomo 2-asalariado 3-empresario 4-Autonomos 5-Asalariados
# estado_civil: 0-soltero 1-casado 2-divorciado
# hijos: Cantidad de hijos menores (no trabajan)
# comprar: 0-mejor alquilar 1-Comprar casa
# hipoteca fija a 30 años con interes
```

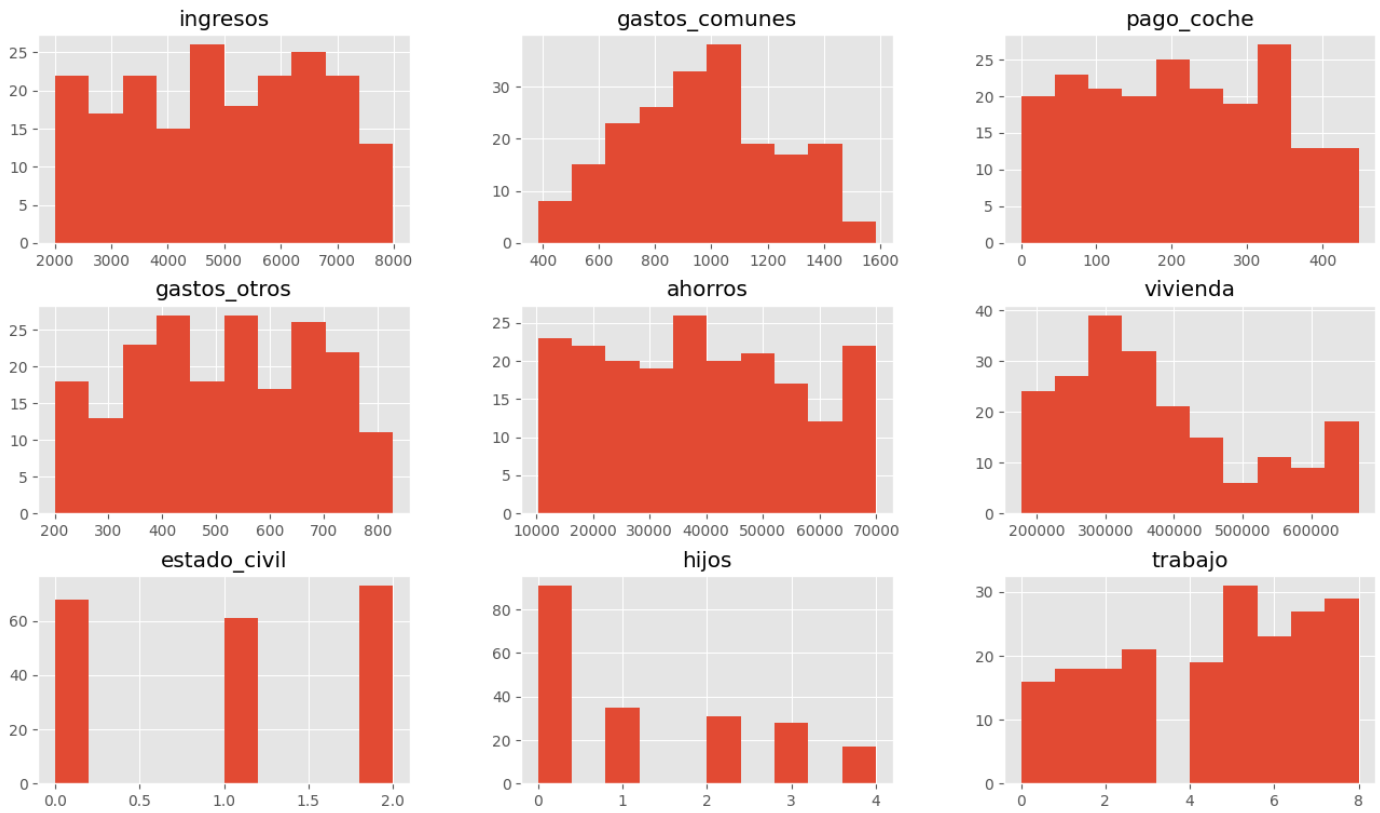
Out[6]:

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	comprar
0	6000	1000	0	600	50000	400000	0	2	2	1
1	6745	944	123	429	43240	636897	1	3	6	0
2	6455	1033	98	795	57463	321779	2	1	8	1
3	7098	1278	15	254	54506	660933	0	0	3	0
4	6167	863	223	520	41512	348932	0	0	3	1
5	5692	911	11	325	50875	360863	1	4	5	1
6	6830	1298	345	309	46761	429812	1	1	5	1
7	6470	1035	39	782	57439	606291	0	0	1	0
8	6251	1250	209	571	50503	291010	0	0	3	1
9	6987	1258	252	245	40611	324098	2	1	7	1

In [7]:
```
print(dataframe.groupby('comprar').size())
```

```
comprar
0    135
1     67
dtype: int64
```

In [8]:
```
dataframe.drop(['comprar'], axis=1).hist()
plt.show()
```



Preprocesamos los datos

Creamos 2 columnas nuevas. En una agrupamos los gastos mensuales. En la otra el monto a financiar para comprar la casa

In [9]:
```
dataframe['gastos']=(dataframe['gastos_comunes']+dataframe['gastos_otros']+dataframe['pa
dataframe['financiar']=dataframe['vivienda']-dataframe['ahorros']
dataframe.drop(['gastos_comunes','gastos_otros','pago_coche'], axis=1).head(10)
```

Out[9]:

	ingresos	ahorros	vivienda	estado_civil	hijos	trabajo	comprar	gastos	financiar
0	6000	50000	400000	0	2	2	1	1600	350000
1	6745	43240	636897	1	3	6	0	1496	593657
2	6455	57463	321779	2	1	8	1	1926	264316
3	7098	54506	660933	0	0	3	0	1547	606427
4	6167	41512	348932	0	0	3	1	1606	307420
5	5692	50875	360863	1	4	5	1	1247	309988
6	6830	46761	429812	1	1	5	1	1952	383051
7	6470	57439	606291	0	0	1	0	1856	548852
8	6251	50503	291010	0	0	3	1	2030	240507
9	6987	40611	324098	2	1	7	1	1755	283487

In [11]:
```
dataframe.shape
```

Out[11]:
```
(202, 12)
```

Información estadística de los datos

In [12]:
```
reduced = dataframe.drop(['gastos_comunes','gastos_otros','pago_coche'], axis=1)
reduced.describe()
```

Out[12]:

	ingresos	ahorros	vivienda	estado_civil	hijos	trabajo	comprar	gastos	
count	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	202.000000	
mean	4958.995050	38749.668317	373349.638614	1.024752	1.232673	4.490099	0.331683	1698.752475	3
std	1682.862556	17365.231870	136371.525622	0.837184	1.367833	2.535794	0.471988	324.838005	1
min	2008.000000	10319.000000	176553.000000	0.000000	0.000000	0.000000	0.000000	1007.000000	1
25%	3513.750000	24964.250000	274810.000000	0.000000	0.000000	2.000000	0.000000	1430.500000	2
50%	4947.500000	38523.000000	340783.500000	1.000000	1.000000	5.000000	0.000000	1669.500000	3
75%	6374.500000	52150.750000	444482.000000	2.000000	2.000000	7.000000	1.000000	1928.000000	3
max	7984.000000	69934.000000	669540.000000	2.000000	4.000000	8.000000	1.000000	2543.000000	6

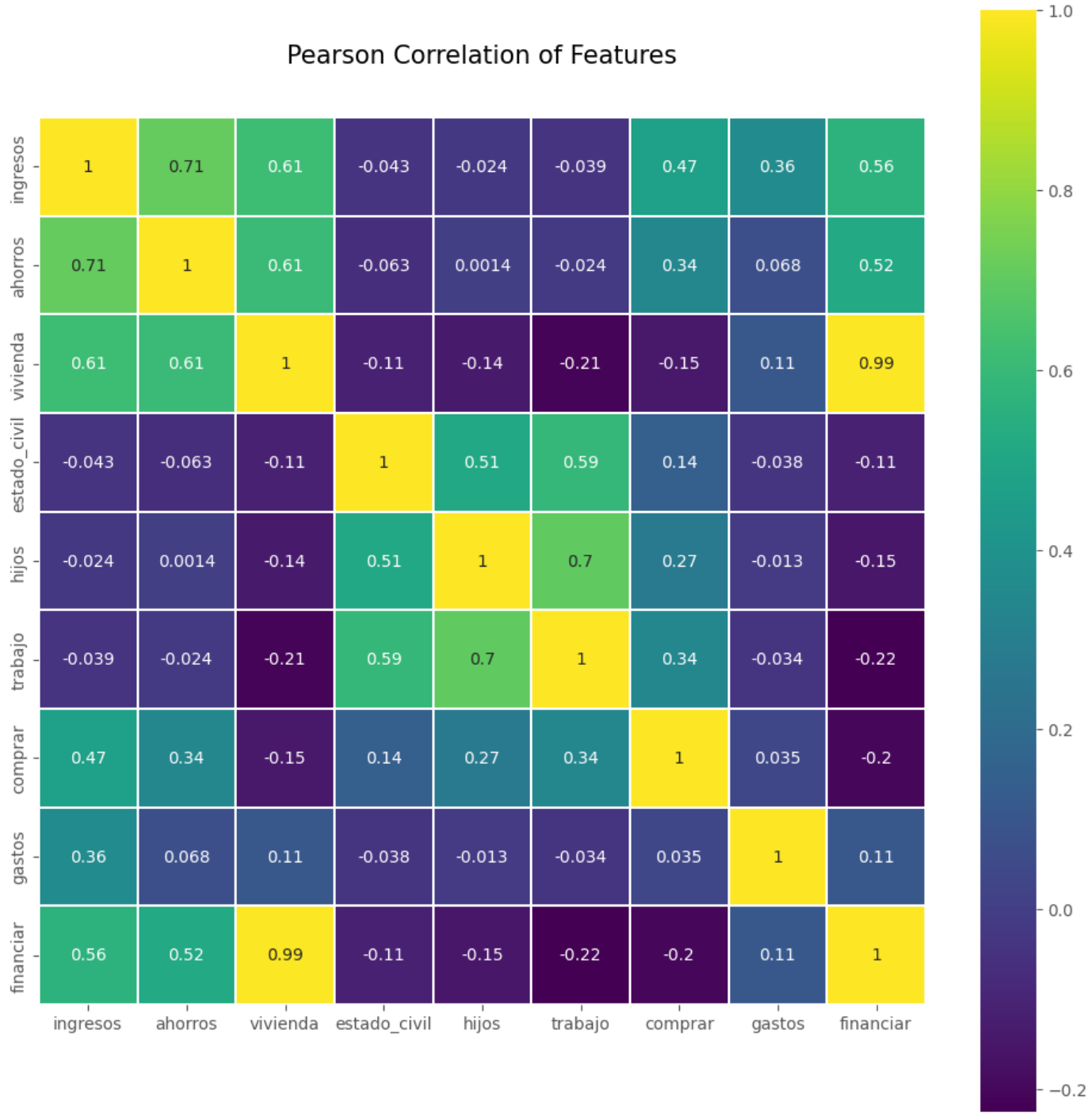
Feature Selection

Selección de las características de entrada.

Veamos la correlación entre columnas

In [13]:
```
colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(reduced.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=color
```

Out[13]:
```
<Axes: title={'center': 'Pearson Correlation of Features'}>
```



Usemos la clase SelectKBest de SkLearn para elegir las 5 mejores características a usar.

In [14]:
```
X=dataframe.drop(['comprar'], axis=1)
y=dataframe['comprar']

best=SelectKBest(k=5)
X_new = best.fit_transform(X, y)
X_new.shape
selected = best.get_support(indices=True)
print(X.columns[selected])
```

```
Index(['ingresos', 'ahorros', 'hijos', 'trabajo', 'financiar'], dtype='object')
```

Veamos la correlación entre las 5 mejores Features

In [15]:
```
used_features =X.columns[selected]

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
```

```
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(dataframe[used_features].astype(float).corr(),linewidths=0.1,vmax=1.0, square
```

Out[15]:

```
<Axes: title={'center': 'Pearson Correlation of Features'}>
```

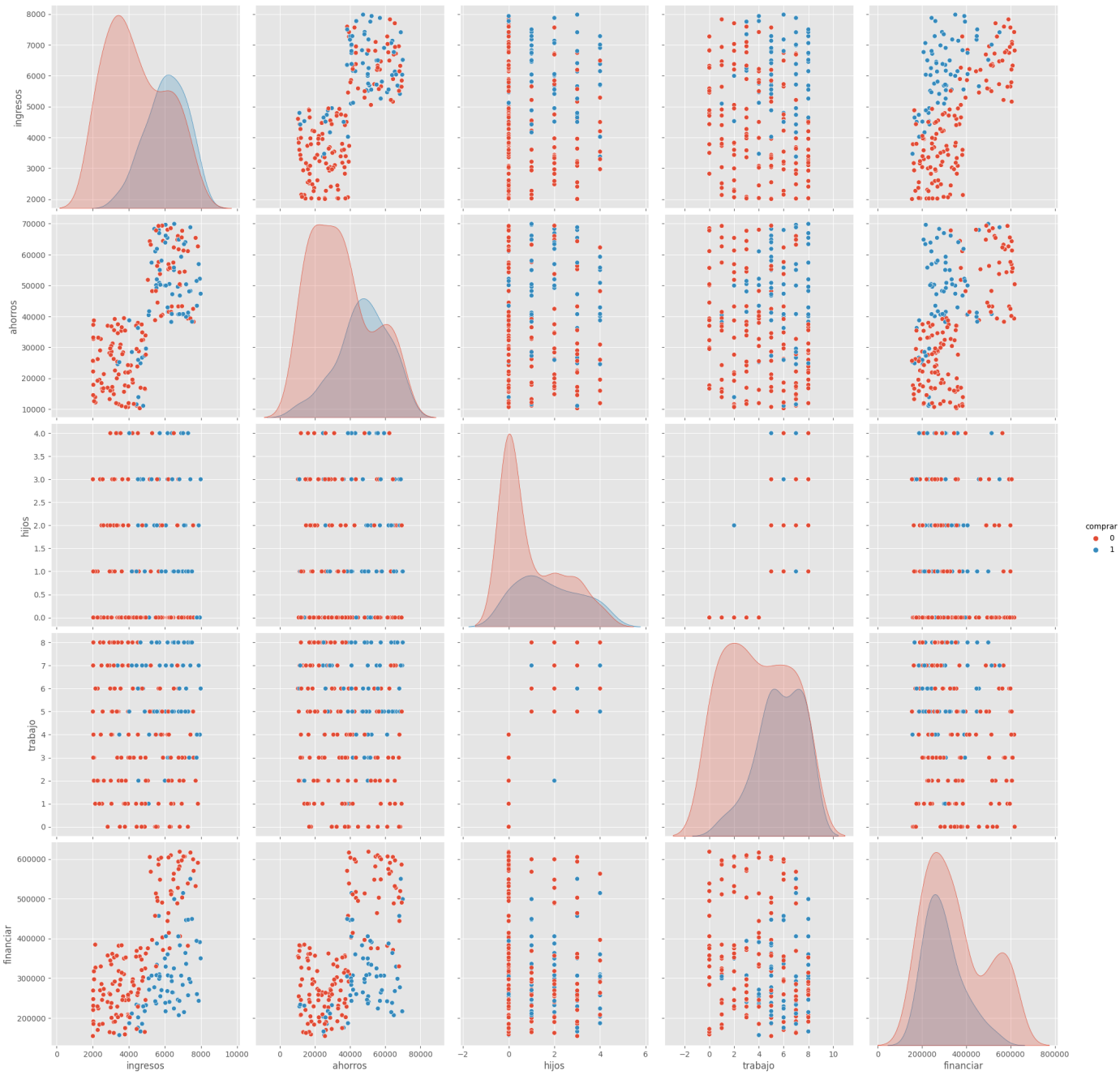
In [16]:

```
sb.pairplot(dataframe, hue='comprar',size=4,vars=used_features,kind='scatter')
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:2095: UserWarning: The `size
` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[16]:

```
<seaborn.axisgrid.PairGrid at 0x7a91cfb2a9b0>
```



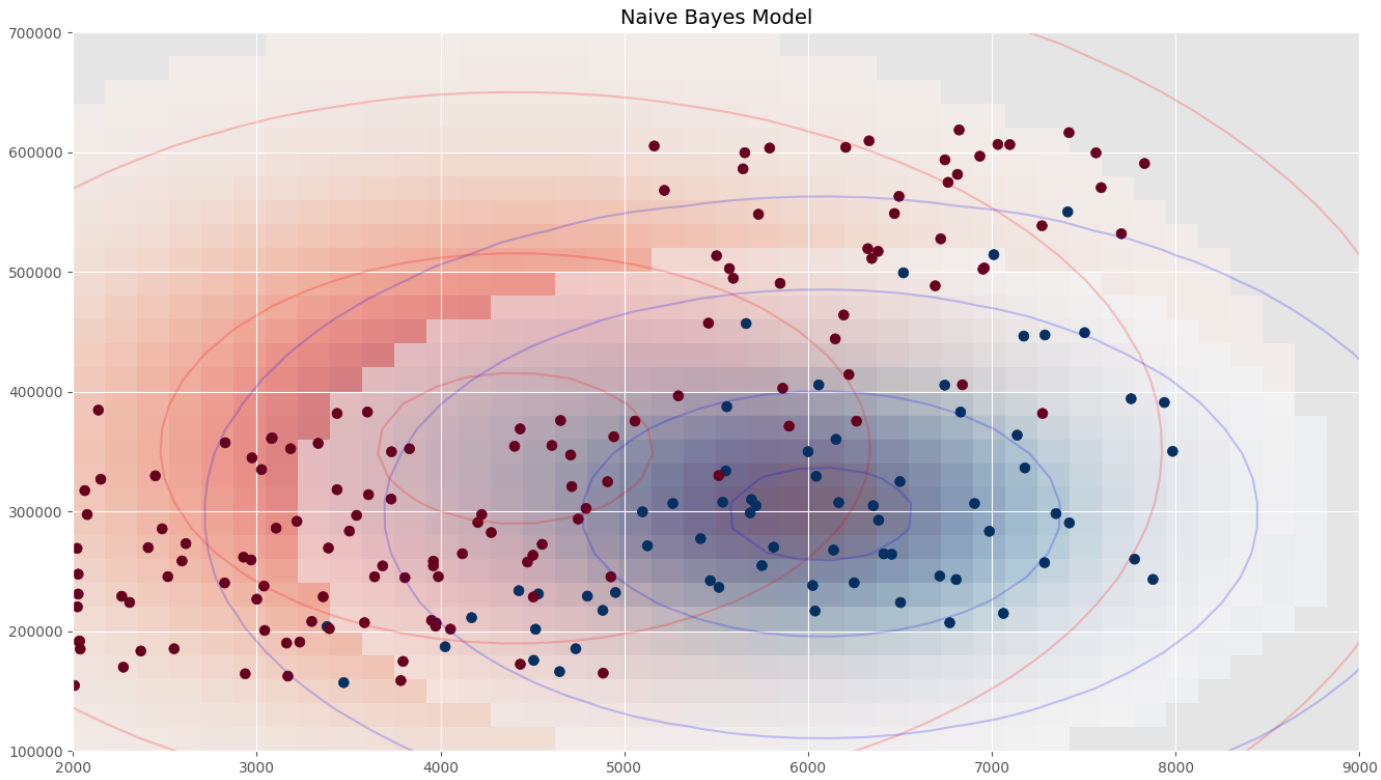
Ejemplo: Graficar 2 de las Características

In [17]:

```
used_features2 =[
    "ingresos",
    "financiar"
]
X=dataframe[used_features2].values
y=dataframe["comprar"]

fig, ax = plt.subplots()

ax.scatter(X[:,0], X[:,1], c=y, s=50, cmap='RdBu')
ax.set_title('Naive Bayes Model', size=14)

xlim = (2000, 9000)
ylim = (100000, 700000)

xg = np.linspace(xlim[0], xlim[1], 40)
yg = np.linspace(ylim[0], ylim[1], 30)
xx, yy = np.meshgrid(xg, yg)
Xgrid = np.vstack([xx.ravel(), yy.ravel()]).T
```

```
for label, color in enumerate(['red', 'blue']):
    mask = (y == label)
    mu, std = X[mask].mean(0), X[mask].std(0)
    P = np.exp(-0.5 * (Xgrid - mu) ** 2 / std ** 2).prod(1)
    Pm = np.ma.masked_array(P, P < 0.03)
    ax.pcolorfast(xg, yg, Pm.reshape(xx.shape), alpha=0.5,
                  cmap=color.title() + 's')
    ax.contour(xx, yy, P.reshape(xx.shape),
               levels=[0.01, 0.1, 0.5, 0.9],
               colors=color, alpha=0.2)

ax.set(xlim=xlim, ylim=ylim)
```

Out[17]:
```
[(2000.0, 9000.0), (100000.0, 700000.0)]
```



Vemos que cuantos más ingresos tiene la familia y menor es la cantidad a financiar, mejora la opción de Comprar (en azul). De otra manera, convendrá alquilar (rojo)

Creamos el modelo de Gaussian Naive Bayes

Dividimos en un set de Entrenamiento y otro de Test con el 20% de las entradas

In [19]:
```
# Split dataset in training and test datasets
X_train, X_test = train_test_split(dataframe, test_size=0.2, random_state=6)
y_train =X_train["comprar"]
y_test = X_test["comprar"]
```

Entrenamos el modelo

In [21]:
```
used_features
```

Out[21]:
```
Index(['ingresos', 'ahorros', 'hijos', 'trabajo', 'financiar'], dtype='object')
```

In [22]:
```
# Instantiate the classifier
gnb = GaussianNB()
```

```
# Train classifier
gnb.fit(
    X_train[used_features].values,
    y_train
)
y_pred = gnb.predict(X_test[used_features])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but GaussianNB was fitted without feature names
  warnings.warn(
```

Resultados

In [23]:
```
print('Precisión en el set de Entrenamiento: {:.2f}'
     .format(gnb.score(X_train[used_features], y_train)))
print('Precisión en el set de Test: {:.2f}'
     .format(gnb.score(X_test[used_features], y_test)))
```

```
Precisión en el set de Entrenamiento: 0.87
Precisión en el set de Test: 0.90
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but GaussianNB was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature
names, but GaussianNB was fitted without feature names
  warnings.warn(
```

In [24]:
```
# Print Test results
print("Total de Muestras en Test: {}\nFallos: {}"
      .format(
          X_test.shape[0],
          (y_test != y_pred).sum()
))
```

```
Total de Muestras en Test: 41
Fallos: 4
```

In [25]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[24  0]
 [ 4 13]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        24
           1       1.00      0.76      0.87        17

    accuracy                           0.90        41
   macro avg       0.93      0.88      0.89        41
weighted avg       0.92      0.90      0.90        41
```

Nuevas Predicciones

Hagamos 2 pruebas a modo de ejemplo en las que el algoritmo nos recomienda Alquilar (0) y Comprar (1) en el 2do caso

In [27]:
```
#                  ['ingresos', 'ahorros', 'hijos', 'trabajo', 'financiar']
print(gnb.predict([[2000,        5000,     0,       5,         200000],
                   [6000,        34000,    2,       5,         320000] ]))
#Resultado esperado 0-Alquilar, 1-Comprar casa
```

```
[0 1]
```

In [ ]: