Tecnológico de Monterrey

Evidencia 1. Portafolio de análisis

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Aylin Camacho Reyes - A01379272 (The cool other)

Ivan Mauricio Amaya Contreras

08 de Septiembre del 2022, Monterrey N.L.

Durante cinco semanas hemos visto como funcionan los modelos de predicción. En este caso se utilizo un conjunto de datos que contenia información sobre los componentes del vino, para el analisis de los datos se usaron frameworks de python que facilitaran el proceso.

Exploración de datos

Esta fase nos ayuda a conocer la información que nos estan brindando, saber si tiene variables culitativas o cuantitativas, como se encuentran distribuidos los datos por medio de la estadistica descriptiva y si existen inconsistencias como valores nulos.

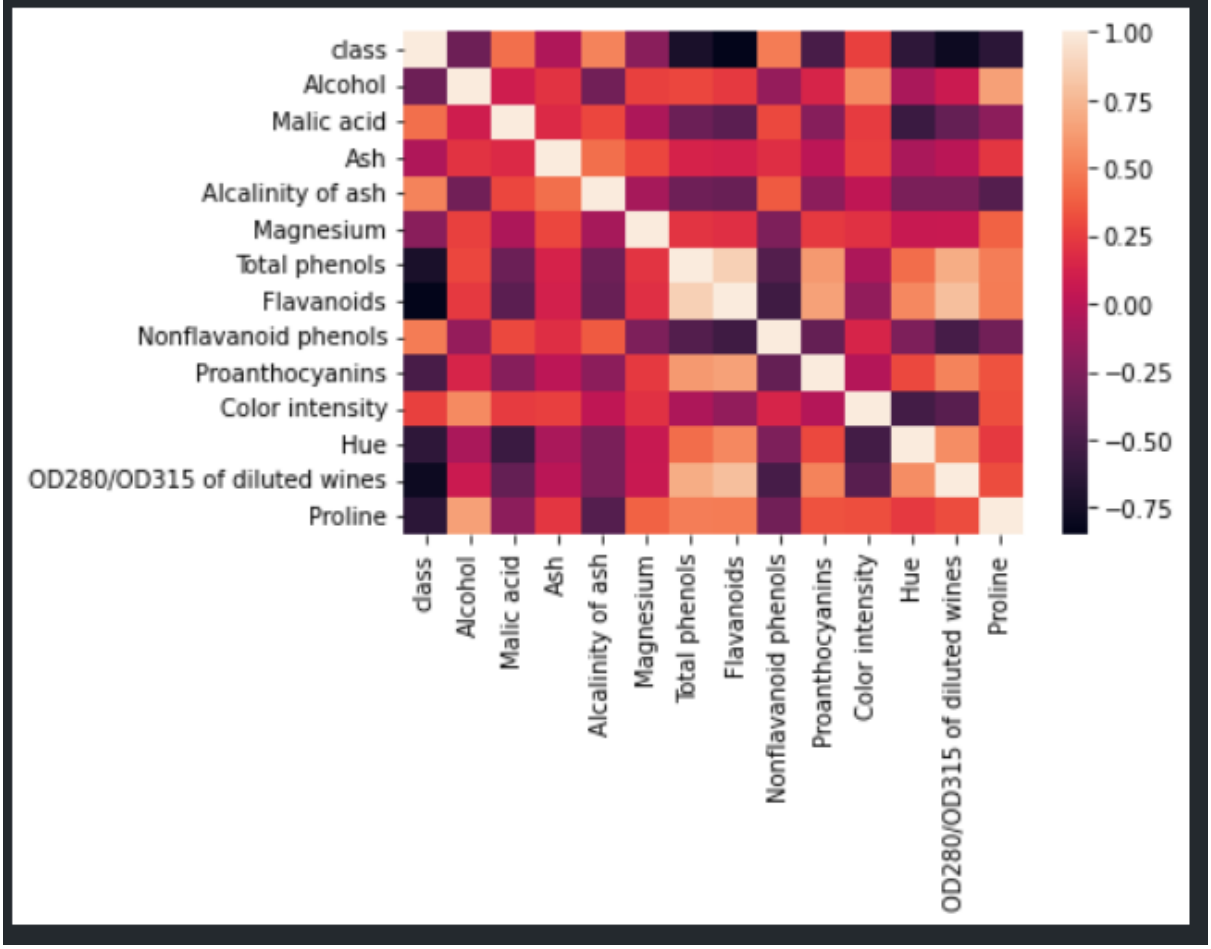
```
#exploring the data

#data types
print(df.info())

#statistical summary
print(df.describe())

#inconsistencies
print(df.isnull().sum())
print(df.isna().sum())
```

Tambien para explorar los datos se realizon un heat map para observar la relación entre las variables con el coeficiente de pearson, en este caso no consideramos que las dos variables que mostraban mayor correlación fueran a ser las unicas que determinaran el valor de la variables dependiente.

```
#Correlacion de variables
corr = df.corr()  # Coeficiente de Pearson
sb.heatmap(corr)
```

✓ 0.6s

```
<AxesSubplot:>
```

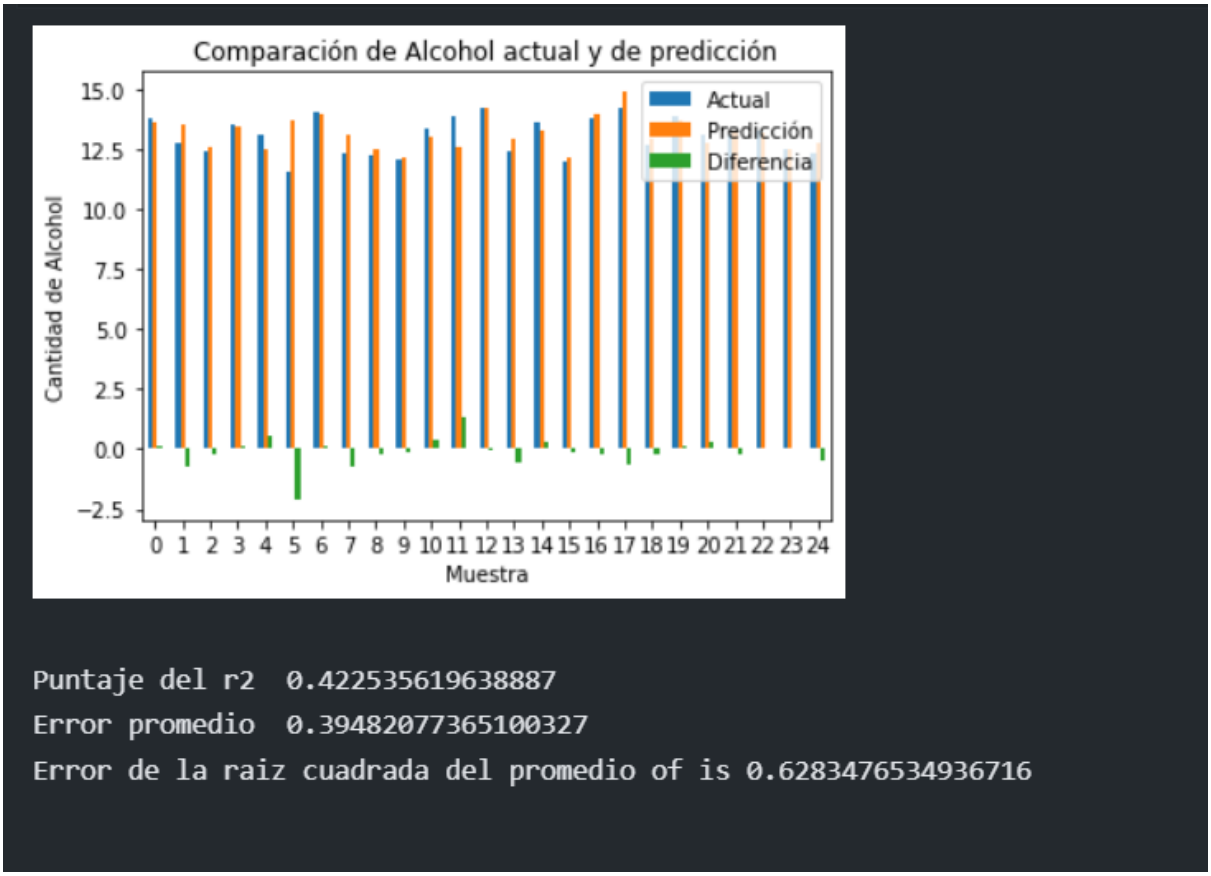


Separación del dataset (Train/Test/Validation)

Una vez entendidos los datos se pason a la parte del modelado. Primero se separa el dataset entre variables dependientes e independientes, en este caso escogimos como variable dependiente el nivel de alcohol del vino y como variables independientes el resto de columnas a excepción de “class” ya que era una variable de tipo cualitativa.

```
#modeling

x = df[["Malic acid", "Ash", "Alcalinity of ash", "Magnesium", "Total phenols",  "Flavanoids",
"Nonflavanoid phenols", "Proanthocyanins", "Color intensity", "Hue", "OD280/OD315 of diluted wines", "Proline"]].values
y = df[['Alcohol']].values
✓ 0.4s

X_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

modelo_regresion = LinearRegression()  # modelo de regresión
```

Para inicar el modelo se declaran las variables independientes como “x” y las independientes como “y”, para separar los datos en los que seran usados para prueba (20%) y los de validación (8%0) con la ayuda de las librerias de sklearns.




```
Puntaje del r2  0.422535619638887
Error promedio  0.39482077365100327
Error de la raiz cuadrada del promedio of is 0.6283476534936716
```

Diagnóstico del grado de bias o sesgo

```
# calculate the bias and variance
bias = np.mean(y_test - y_pred)
print('Bias: %f' % bias)
```

✓ 0.7s

```
Bias: -0.057746
```

El sesgo negativo nos indica que gran parte de los datos estan por debajo de la media, por lo que se encuentran por debajo del valor real.

Diagnóstico sobre el grado de varianza

```
variance = np.mean((y_test - y_pred)**2
print('Variance: %f' % variance)
```

✓ 0.5s

```
Variance: 0.394821
```

Este valor nos indica que tanto cambiaran los valores de predicción si se cambian los datos de entrenamiento, este valor no es optimo, pero se puede mejorar

Diagnóstico sobre el nivel de ajuste del modelo

El modelo es overfitting ya que la varianza es alta, en este caso lo que se tendría que hacer para mejorar las predicciones del modelo es buscar disminuir la varianza para aumentar el bias o vicebersa.