Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Puebla

Desarrollo de proyectos de análisis de datos

IN1002B, Grupo 303

Nombre del profesor:
Alfredo García

Entregable: Actividad 4 Reporte

Equipo | Integrantes:

José Gonzalo Varela Zarate

Saúl Alberto Sánchez Cervantes

Hiram Jimenez Cabrera

Carlos Santiago Lamas Camacho

Lizeth Guadalupe Mejía Morales

Harry Hernández Grande

Noviembre 2022

En esta práctica abordaremos los pasos que seguimos para el análisis del archivo *"microretailer_mit_lift_lab_actualizado.csv*." primeramente se realizó como ya conocemos la importación de las distintas librerías como numpy, matplot, pandas, seaborn, etc. proseguimos a la eliminación de valores nulos y a reemplazarlos, a continuación una descripción del método utilizado:

```
#Reemplazamos valores nulos con el método más sencillo
data2_Micro_Retailer=Micro_Retailer.fillna(method='bfill')

#Reemplazamos valores nulos con el método más sencillo
data2_Micro_Retailer=data2_Micro_Retailer.fillna(method='ffill')
data2_Micro_Retailer
```

Una vez logrado eso se identificaron y eliminaron los outliers mediante el método de intercuartiles después de haber separado únicamente las variables cualitativas de las cuantitativas ya que son únicamente estas últimas las que nos interesa analizar.

```
#Filtro por columnas (Usamos solo las columnas de variables
cuantitativas)
filtro_columnas_innecesarias=data2_Micro_Retailer.iloc[ : ,
[5,6,13,14,16,21,22,31,42,49,51,59,29,33,78,90]]
filtro_columnas_innecesarias
```

```
#Método aplicando cuartiles Encuentro cuartules de 0.25 y 0.75
y=filtro_columnas_innecesarias
percentile25=y.quantile(0.25) #Q1
percentile75=y.quantile(0.75) #Q3
iqr= percentile75 - percentile25

Limite_Superior_iqr= percentile75 + 1.5*iqr
Limite_Inferior_iqr= percentile25 - 1.5*iqr
print("Limite superior permitido", Limite_Superior_iqr)
print("Limite inferior permitido", Limite_Inferior_iqr)
```

```
#Obtenemos datos limpios del dataframe
data3_iqr_Micro_Retailer=y[(y<=Limite_Superior_iqr)&(y>=Limite_Inferior
_iqr)]
data3_iqr_Micro_Retailer
```

Una vez que tenemos el dataframe limpion de outliers y nulos procedimos a observar las variables del dataframe y realizar un análisis de correlación entre distintas variables, empezando con lo mencionado justo abajo.

```
#analizamos caso 1 de correlación entre variable de número de clientes en la tienda respecto al número de estantes en la tienda, el número de refrigeradores y el número de empleados permanentes
Vars_Indep=
data4_iqr[['268_number_fridges','104_how_many_shelves_does_the_micro_retailer_have', '2_current_permanent_employees']]
Var_Dep= data4_iqr['97_number_of_customers_in_store']
```

A continuación se define una función de regresión lineal, ajustamos el modelo a las variables determinadas, verificamos los coeficientes y evaluamos la eficiencia del modelo mediante el R obtenido, en este caso es un modelo bastante aceptable con un R determinado en "*0.1130725423688903*"

```
#Evaluamos la eficiencia del modelo obtenido por medio del coeficiente R determinado
model.score(Vars_Indep,Var_Dep)
```

Y cabe destacar que los coeficientes obtenidos son los siguientes

```
{'fit_intercept': True,
 'normalize': 'deprecated',
 'copy_X': True,
 'n_jobs': None,
 'positive': False,
 'feature_names_in_': array(['268_number_fridges',
        '104_how_many_shelves_does_the_micro_retailer_have',
        '2_current_permanent_employees'], dtype=object),
 'n_features_in_': 3,
 'coef_': array([0.19590676, 0.0080384 , 0.08928311]),
 '_residues': 972.3842620033486,
 'rank_': 3,
 'singular_': array([260.78872511,  58.823247  ,  42.76081943]),
 'intercept_': 0.8414396869333318}
```

Para la regresión logística procedimos a definir las variables dependientes e independientes para luego dividir el conjunto de datos de entrenamiento y prueba, escalamos los datos y definimos el algoritmo siguiente

```
#Definimos el algoritmo a utilizar
from sklearn.linear_model import LogisticRegression
algoritmo = LogisticRegression()
```

Después entrenamos al modelo, realizamos una predicción y verificamos la matriz de confusión para después conseguir calcular la exactitud, precisión y sensibilidad del modelo como es mostrado abajo

```
#Verifico la matriz de Confusión
from sklearn.metrics import confusion_matrix
matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)
```

```
Matriz de Confusión:
[[45  1  0  0  0  0  0  0  0]
 [16  0  1  0  0  0  0  0  0]
 [16  0  2  2  0  0  0  0  0]
 [ 8  0  0  0  0  0  0  0  0]
 [ 3  0  0  0  0  0  0  0  0]
 [ 2  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  0  0]
 [ 0  0  1  0  0  0  0  0  0]]
```

```
#Calculo la exactitud del modelo
from sklearn.metrics import accuracy_score
exactitud= accuracy_score(y_test, y_pred)
print('Exactitud del modelo')
print(exactitud)
```

0.47474747474747475

```
#Calculo la precisión del modelo
from sklearn.metrics import precision_score
precision= precision_score(y_test, y_pred, average="micro",
pos_label="yes")
print('Precision del modelo:')
print(precision)
```

0.47474747474747475

```
#Calculo la sensibilidad del modelo
from sklearn.metrics import recall_score
sensibilidad=recall_score(y_test, y_pred, average='micro',
pos_label="yes")
print('Sensibilidad del modelo')
print(sensibilidad)
```

0.47474747474747475

Concluyendo podemos determinar que el cálculo es medianamente adecuado a lo que predecimos y modelamos sin embargo quedamos con ciertas dudas y unos cuantos defectos que seguiremos mejorando para lograr un modelo mucho más exacto y preciso y que de ese modo el proyecto salga a flote con óptima funcionalidad.