1. Exploración de datos

En [39]:

```
#Importamos librerías necesarias
import  pandas  como  pd
import  numpy  como  np
import  matplotlib.pyplot  como  plt
import  seaborn  como  sns
```

En [40]:

```
#Para ignorar los avisos en el código
import  avisos
avisos . advertencias de filtro ( "ignorar" )
```

En [41]:

```
#Creando función que permite leer el conjunto de datos
desde  type_extensions  import  dataclass_transform
def  archivo ( ubicación , archivo , extensión ):
  if  extensión  == ".csv" :
    datos  =  pd . read_csv ( ubicación + archivo + extensión )
    devolver  datos

   extensión  elif ==  ".xlsx" :
    datos  =  pd . read_excel ( ubicación + archivo + extensión )
    devolver  datos

   extensión  elif == ".json" :
    datos  =  pd . read_json ( ubicación + archivo + extensión )
    devuelve  datos
  else :
    print ( "Extensión no valida" )
```

En [42]:

```
#Asiganamos a la variable data nuestro dataset
data  =  archivo ( "/content/" , "petrol_consumption" , ".csv" )
data . head ( 10 )  #Mostramos las 10 primeras filas
```

Fuera[42]:

	Impuesto sobre la gasolina	Ingresos promedio	Carreteras pavimentadas	Población_licencia_de_conducir(%)	Consumo de gasolina
0	9.0	3571	1976	0,525	541
1	9.0	4092	1250	0,572	524
2	9.0	3865	1586	0,580	561
3	7.5	4870	2351	0,529	414
4	8.0	4399	431	0,544	410
5	10.0	5342	1333	0,571	457
6	8.0	5319	11868	0,451	344
7	8.0	5126	2138	0,553	467
8	8.0	4447	8577	0,529	464
9	7.0	4512	8507	0,552	498

En [43]:

```
#Información del conjunto de
datos . información ()
```

```
<clase 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entradas, 0 a 47
Columnas de datos (5 columnas en total):
 # Columna Conteo no nulo Dtype
--- ------ -------------- -----
 0 Impuesto a la gasolina 48 float no nulo64
 1   Average_income                48 non-null     int64
 2   Paved_Highways                48 non-null     int64
 3   Population_Driver_licence(%)  48 non-null     float64
 4   Petrol_Consumption            48 non-null     int64
dtypes: float64(2), int64(3)
memory usage: 2.0 KB
```

In [44]:

```
#Filas y columnas del dataset
data.shape
```

Out[44]:

```
(48, 5)
```

In [45]:

```
#Miramos los valores NaN del dataset
data.isna().sum()
```

Out[45]:

```
Petrol_tax                      0
Average_income                  0
Paved_Highways                  0
Population_Driver_licence(%)    0
Petrol_Consumption              0
dtype: int64
```

In [46]:

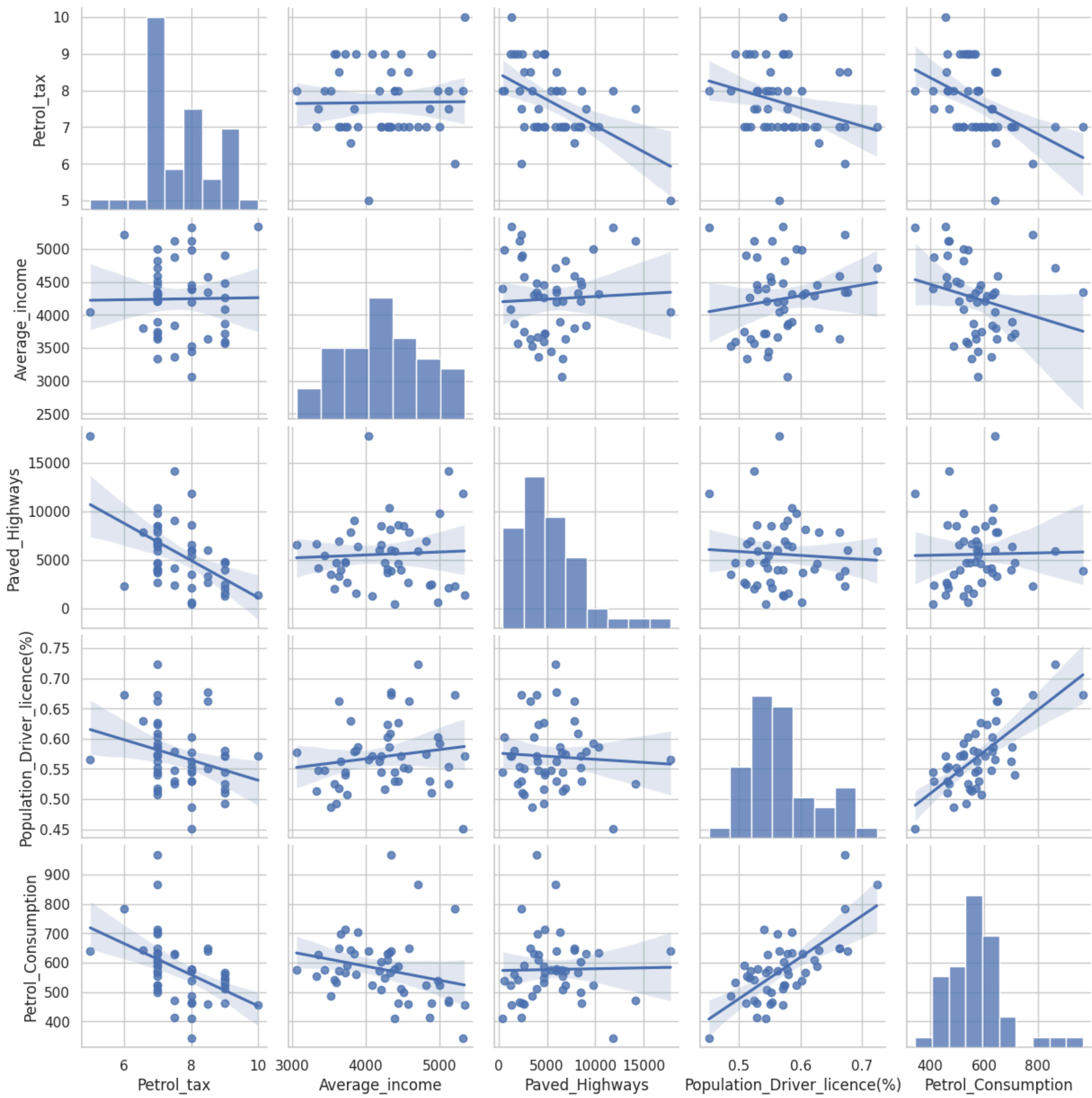
```
#Mostramos datos estadisticos del dataset
data.describe().T
```

Out[46]:

	count	mean	std	min	25%	50%	
Petrol_tax	48.0	7.668333	0.950770	5.000	7.00000	7.5000	8.1
Average_income	48.0	4241.833333	573.623768	3063.000	3739.00000	4298.0000	4578.7
Paved_Highways	48.0	5565.416667	3491.507166	431.000	3110.25000	4735.5000	7156.0
Population_Driver_licence(%)	48.0	0.570333	0.055470	0.451	0.52975	0.5645	0.5
Petrol_Consumption	48.0	576.770833	111.885816	344.000	509.50000	568.5000	632.7

In [47]:

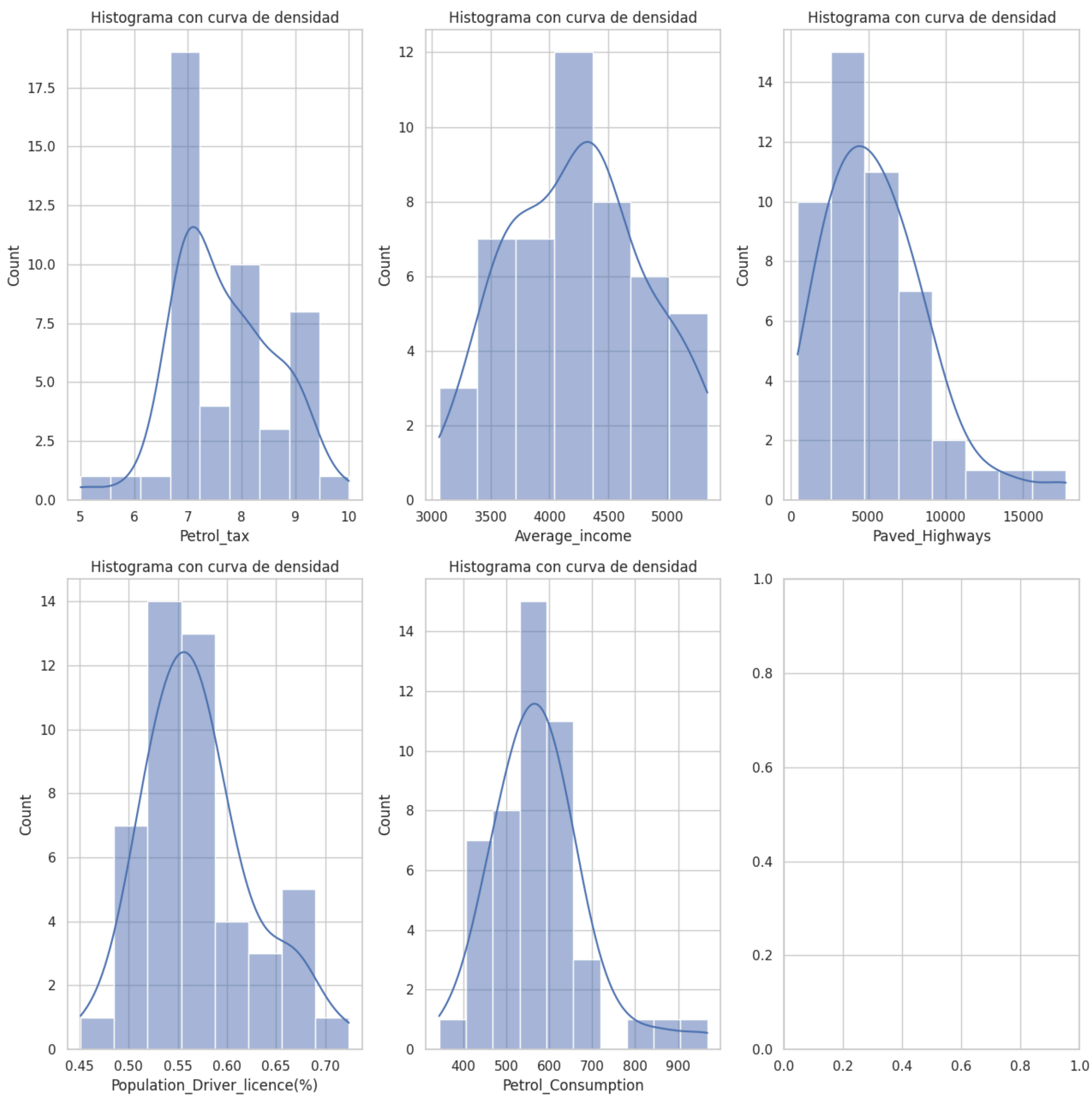
```python
# Gráfico con regresión lineal
sns.pairplot(data, kind='reg')
plt.show()
```



Vemos que existe una relación lineal entre algunas variables, eso significa que se puede utlizar un modelo de regresion lineal.

In [48]:

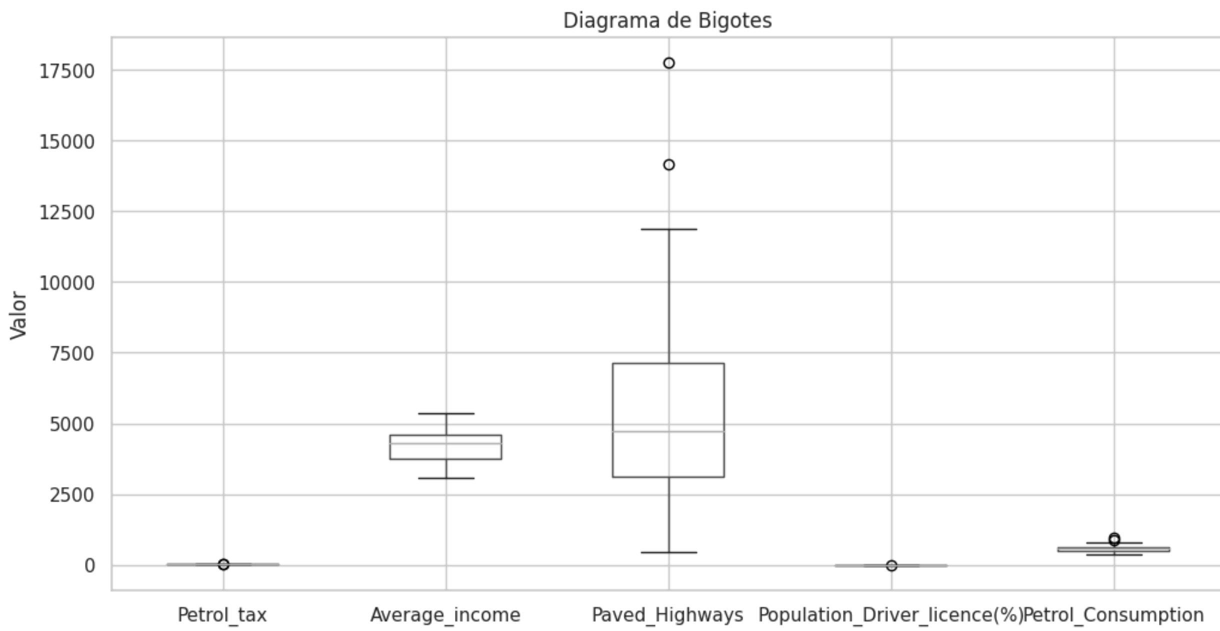
```python
#Creamos grafico de barras con su densidad la cual permite ver la distribución de los

#Asignamos las columnas del dataset a la variable "columnas"
columnas = ["Petrol_tax","Average_income","Paved_Highways","Population_Driver_licence(

fig, axes = plt.subplots(nrows=2,ncols=3,figsize=(13,13))#nrows:# filas - ncols:#colum
axes = axes.flatten()

for i, var in enumerate(columnas):
  ax = axes[i]
  sns.histplot(data[var],kde=True,ax=ax)
  ax.set_title("Histograma con curva de densidad")
  ax.set_xlabel(var)

plt.tight_layout()
plt.show()
```



Vemos que ninguna densidad de las variables tiene forma de campana Gaussiana, lo que significa que toca normalizar la data.

In [49]:
```
#Asignamos las columnas del dataset a la variable "columnas"
columnas = ["Petrol_tax", "Average_income", "Paved_Highways", "Population_Driver_licer

#Creamos diagrama de bigotes para ver los datos atipicos del dataset de cada columna
data[columnas].boxplot(figsize=(12, 6))
plt.title('Diagrama de Bigotes')
plt.ylabel('Valor')
```

Out[49]:
```
Text(0, 0.5, 'Valor')
```



Notamos que existe pocos datos atipicos en la data, lo que significa que existe poco ruido y nos indica que toca normalizar la data.

Normalización

In [50]:
```
#Datos estadisticos del dataset con la transpuesta
data.describe().T
```

Out[50]:

	count	mean	std	min	25%	50%	
Petrol_tax	48.0	7.668333	0.950770	5.000	7.00000	7.5000	8.1
Average_income	48.0	4241.833333	573.623768	3063.000	3739.00000	4298.0000	4578.7
Paved_Highways	48.0	5565.416667	3491.507166	431.000	3110.25000	4735.5000	7156.0
Population_Driver_licence(%)	48.0	0.570333	0.055470	0.451	0.52975	0.5645	0.5
Petrol_Consumption	48.0	576.770833	111.885816	344.000	509.50000	568.5000	632.7

Se aplica normalizacion porque cumple con estos requisitos:

1. La desviacion estandar estan muy apartadas una de otra.
2. Cuando las variables **no** son normales (Cuado tienen las curvas muy torcidas en la grafica), no estan de forma de campana Gaussiana

3. Hay poco ruido en el dataframe

Se usa la función de normalización **MinMaxScaler**

In [51]:

```
#Funcion normalizacion (MinMaxScaler)

#Importamos la libreria
from sklearn.preprocessing import MinMaxScaler

#Sacamos los valores sin escabezado
def datanormalizados(df):
  valores = df.values
  scaler = MinMaxScaler(feature_range = (0,1)) #Porque la normalizacion va de 0-1
  #Aplica la normalizacion
  scaler = scaler.fit(valores)


#np.vstack = Empieza a unir los valores entre un min y un max para que no se salga del
  pd.DataFrame(np.vstack((scaler.data_min_,scaler.data_max_)),
               index =["Min","Max"],
               columns = df.columns)
#Datos ya normalizados
  normalizados = scaler.transform(valores)
  df_norm = pd.DataFrame(normalizados,index=df.index,columns = df.columns)

  return df_norm
```

In [52]:

```
#Pasando completo el dataset

#Asigna a variable los datos normalizados
datanormal = datanormalizados(data)

#Muestra los valores estadisticos de la data
datanormal.describe().T
```

Out[52]:

	count	mean	std	min	25%	50%	75%	max
Petrol_tax	48.0	0.533667	0.190154	0.0	0.400000	0.500000	0.625000	1.0
Average_income	48.0	0.517259	0.251700	0.0	0.296621	0.541904	0.665094	1.0
Paved_Highways	48.0	0.295915	0.201228	0.0	0.154415	0.248084	0.387586	1.0
Population_Driver_licence(%)	48.0	0.437118	0.203188	0.0	0.288462	0.415751	0.528388	1.0
Petrol_Consumption	48.0	0.373030	0.179304	0.0	0.265224	0.359776	0.462740	1.0

Test de normalidad de shapirowik

Se realiza test de normalidad para definir que variable es más optima para poder predecir el modelo.

In [53]:
```
#Importamos la libreria
from scipy import stats

def testshapirowilk(df):
  valoresp=[]
  concepto=[]
  variable=[]

  for column in df:
    k2,p_value = stats.shapiro(df[column].values)
    valoresp.append(p_value)
    variable.append(column)

    if(p_value < 0.05):
      concepto.append("No es una variable normal")
    else:
      concepto.append("Es una variable normal")

  dfshapiro = pd.DataFrame({"Variable":variable,"Valores P":valoresp,"Concepto":concep
  return dfshapiro

#Muestra resultado
testshapirowilk(data)
```

Out[53]:

	Variable	Valores P	Concepto
0	Petrol_tax	0.001327	No es una variable normal
1	Average_income	0.398738	Es una variable normal
2	Paved_Highways	0.004471	No es una variable normal
3	Population_Driver_licence(%)	0.110676	Es una variable normal
4	Petrol_Consumption	0.009222	No es una variable normal

Se evidencia que solo dos variables son variables normales, lo que significa que con alguna de las 2 podemos hacer un tema de predicción.

En este caso vamos a tomar la variable **Population_Driver_licence(%)** como nuestra variable dependiente y el resto como variables independientes.

Correlación

In [54]:
```
#Aplicando correlacion con metodo spearman
# La correlación se deja con el metodo spearman porque genera correlaciones altas
import seaborn as sns

def diagrama (df,tamuno,tamdos):
  f, ax = plt.subplots(figsize=(tamuno,tamdos))
  sns.heatmap(df.corr(method="spearman"),annot=True, linewidths=5, ax=ax)

diagrama(data,10,10)
```

Se puede evidenciar que existe correlación entre las variables **Petrol_Consumption** - **Population_Driver_Licence(%)** con un 60%, lo que significa que es una correlación regular.

In [55]:
```
#Exportamos data limpia en un nuevo dataset en formato .csv
data.to_csv("petrol_consumption.csv", index = False)
datanormal.to_csv("petrol_consumptionLimpia.csv", index = False)
```

---

2. Aplicamos regresion lineal multiple

La aplicamos porque tenemos multiples variables independientes.Lo que se busca con la regesion lineal multiple es que el modelo encuentre una relación entre las variables independientes con nuestra variable dependiente **Population_Driver_licence(%)**

In [56]:
```python
# Mostrar 5 primera filas del dataset
data.head()
```

Out[56]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

---

3. Revisión de homoscedasticidad

Se aplica el test de breush pagan para evidenciar si nuestros datos tienen homocesdasticidad o heterocedasticidad.

- Si p-value < 0.05 quiere decir que se evidencia heteroscedasticidad
- Si p-value > 0.05 quiere decir que se evidencia homoscedasticidad

Siempre se debe buscar que nuestros datos tengan homoscedasticidad, que los datos esten bien dispersos para que no afecte nuestras predicciones.

In [57]:
```python
#Importamos libreria
import statsmodels .api as sm
from statsmodels.stats.diagnostic import het_breuschpagan
#Definimos X,y
X = datanormal.drop("Population_Driver_licence(%)",axis=1)# Variables independientes
X = sm.add_constant(X) # Añade una constante al modelo
y = datanormal["Population_Driver_licence(%)"]#  Variable dependiente
#Entrenando modelo
modelo = sm.OLS(y,X).fit()

# Se calcula el valor de p-value
_,pvalue,_,_ = het_breuschpagan(modelo.resid,modelo.model.exog)

print("p-value: ",pvalue)
```

```
p-value:  0.17607320260179554
```

Como **(p-value)** esta por encima de 0.05 se considera que tiene homoscedaticidad eso significa que sus datos estan dispersos lo que es bastante bueno.

---

4. Revisión de multicolinealidad

Se aplica metrica de inflación de la varianza (VIF) que nos ayuda si existe multicolinealidad en los datos.

- Si VIF = 1 significa que no existe correlación
- Si VIF esta entre el rango de (1-5) significa que existe una correlación moderada (nada grave)
- Si VIF > 5 significa que son niveles criticos de multicolinealidad

Lo que se busca es que la multicolinealidad no sobrepase el valor de 5 porque esto afectaria la presición, la cual seria menos confiable y generaria inestabilidad en nuestro modelo.

In [58]:
```
#!pip install statsmodels
```

In [59]:
```
#Importa libreria
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Toma estructura de un DataFrame
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns # Toma variable X
# Calcula la inflación de la varianza(VIF)
vif_data["Vif"] = [variance_inflation_factor(X.values,i) for i in range(X.shape[1])]

print(vif_data)
```

```
              Feature        Vif
0               const  56.398535
1          Petrol_tax   1.896274
2      Average_income   1.077678
3      Paved_Highways   1.496108
4  Petrol_Consumption   1.465767
```

Se evidencia que ninguna variable presenta un caso alto de multicolinealidad porque casi todas estan en un rango de 1-2 lo cual significa que es bueno.

---

5. Entrenando modelo de regresión lineal multiple

5.1. 40% test 60% train

In [60]:
```
#Importan librerias necesarias para entrenar nuestro modelo
from sklearn.model_selection import train_test_split #Modelo de entrenamiento y testeo
from sklearn.linear_model import LinearRegression #Modelo de regresion lineal
from sklearn.metrics import mean_squared_error,r2_score #Metricas
```

In [61]:
```
#Seleccionamos nuestros datos en X,y
X = datanormal.drop("Population_Driver_licence(%)",axis=1) # Variables independientes
y = datanormal["Population_Driver_licence(%)"] # Variable dependiente
```

```python
#Entrenamos modelo y realizamos partición de datos para testear y entrenar

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.4, random_state = 6

#Crando modelo de regresión lineal
modelo = LinearRegression()
modelo.fit(X_train,y_train)

#Predicción de los datos de testeo
y_pred = modelo.predict (X_test)

#Calculamos metricas
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2 = round(r2_score(y_test,y_pred),3)

print("Error cuadratico medio (MSE) : ",mse)
print("Error Cuadratico Medio (RMSE) : ",rmse)
print("Coeficiente de Determminacion (R2): ",r2)
```

```
Error cuadratico medio (MSE) :  0.01957633940430175
Error Cuadratico Medio (RMSE) :  0.13991547235492488
Coeficiente de Determminacion (R2):  0.639
```

eficiencia de prediccion del 55%

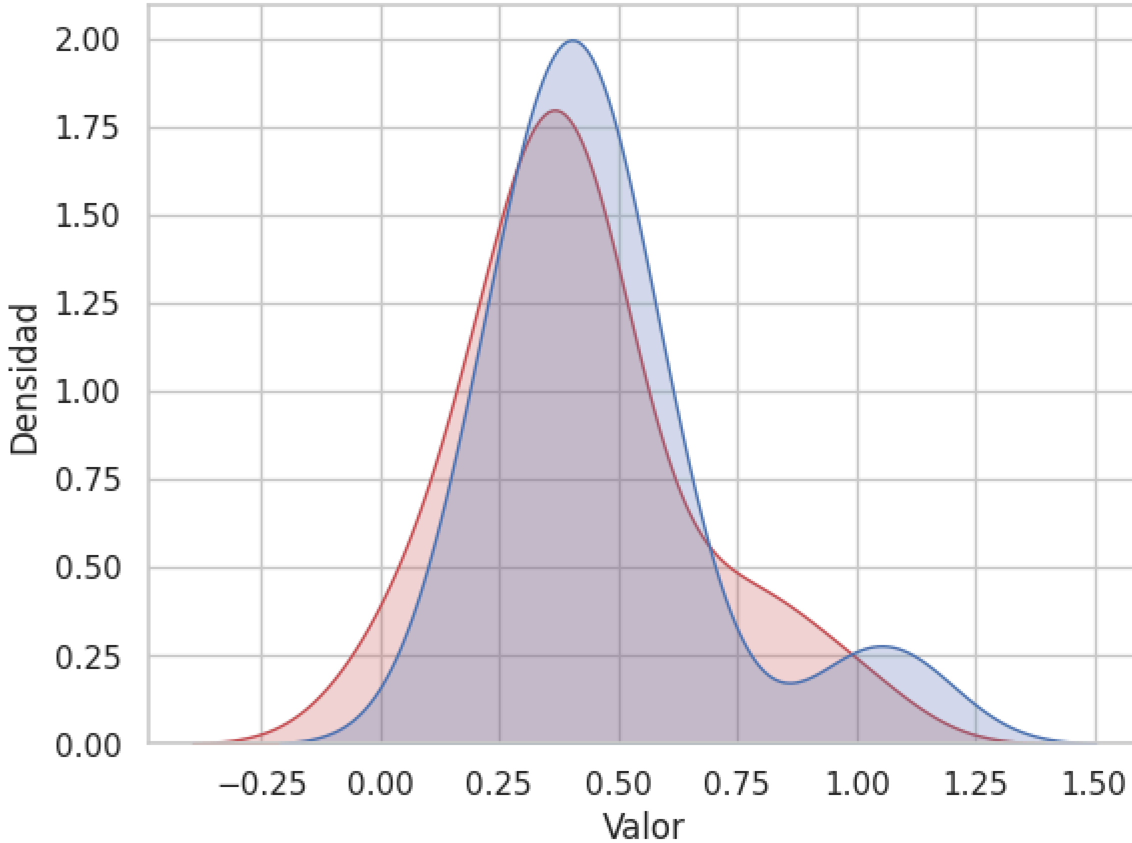
GRAFICA DEL MODELO

In [62]:

```python
import seaborn as sns
ax= sns.kdeplot(y_test,color="r",label="Valores actuales",fill = True)
sns.kdeplot(y_pred,color="b", label="Valores predecidos", fill = True)

plt.xlabel("Valor")
plt.ylabel("Densidad")
```

Out[62]:

```
Text(0, 0.5, 'Densidad')
```

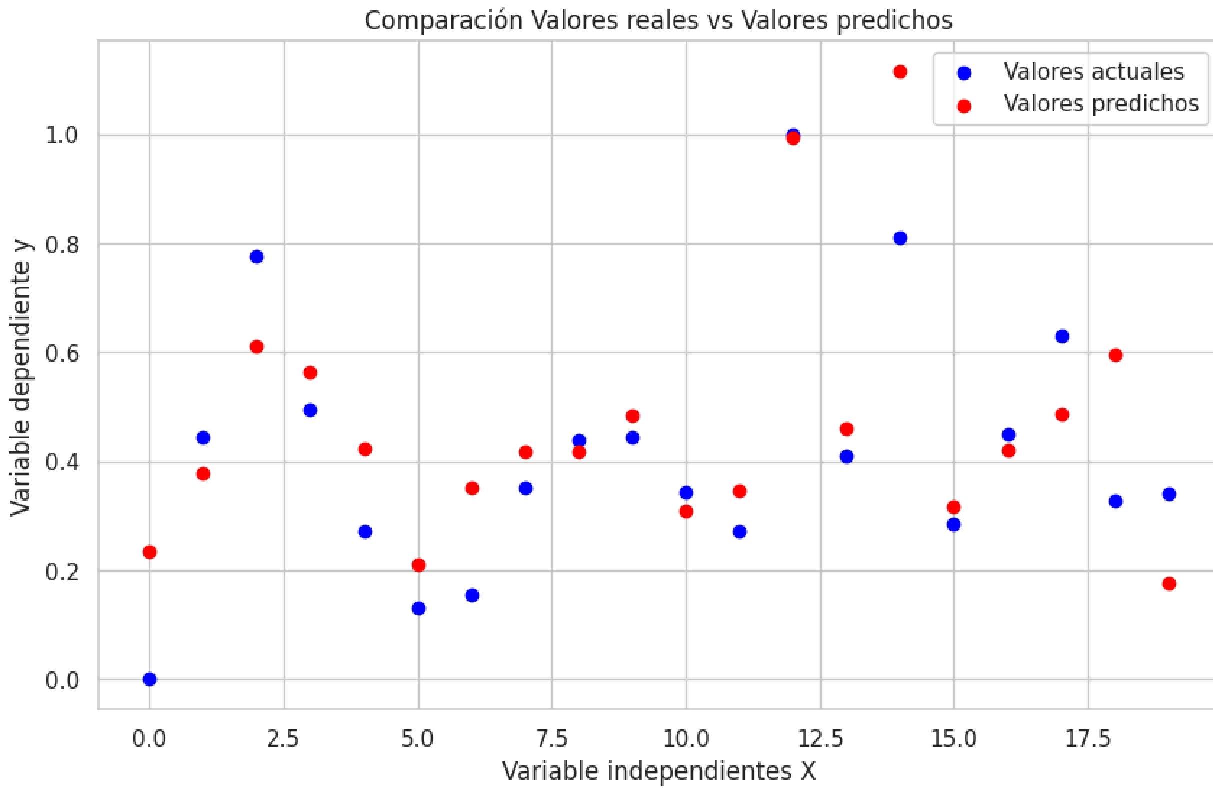


Vemos como los datos reales intentan ajustarse a la curva de los valores predichos

Se evidencia que hay una buena predicción

Comparacion datos actuales con los predichos

In [63]:
```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', label='Valores actuales')
plt.scatter(range(len(y_pred)), y_pred, color='red', label='Valores predichos')
plt.xlabel('Variable independientes X')
plt.ylabel('Variable dependiente y')
plt.title('Comparación Valores reales vs Valores predichos')
plt.legend()
plt.show()
```



Aplicando OLS

In [64]:
```
X =X_test
X= sm.add_constant(X)
y=y_test

modelo_ols = sm.OLS(y,X).fit(cov_type="HC3")
print(modelo_ols.summary())
```

```
                                     OLS Regression Results
=====================================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                       0.
811
Model:                                      OLS   Adj. R-squared:                  0.
760
Method:                           Least Squares   F-statistic:                     7.
619
Date:                          Sat, 04 May 2024   Prob (F-statistic):            0.00
147
Time:                                  16:46:02   Log-Likelihood:                 17.
400
No. Observations:                            20   AIC:                            -2
4.80
Df Residuals:                                15   BIC:                            -1
9.82
Df Model:                                     4
Covariance Type:                            HC3
=====================================================================================
=
                         coef    std err          z      P>|z|      [0.025      0.97
5]
-------------------------------------------------------------------------------------
-
const                  0.1248      0.242      0.516      0.606      -0.349       0.59
9
Petrol_tax            -0.2289      0.202     -1.136      0.256      -0.624       0.16
6
Average_income         0.5271      0.195      2.705      0.007       0.145       0.90
9
Paved_Highways        -0.4598      0.142     -3.243      0.001      -0.738      -0.18
2
Petrol_Consumption     0.7617      0.289      2.637      0.008       0.196       1.32
8
==============================================================================
Omnibus:                        2.190   Durbin-Watson:                   1.807
Prob(Omnibus):                  0.334   Jarque-Bera (JB):                1.018
Skew:                          -0.537   Prob(JB):                        0.601
Kurtosis:                       3.263   Cond. No.                         17.5
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

- %60 train %40 test = 0.854 - 130 -- 0.807 - 123 --- 0.811 - 67
- %70 train %30 test = 0.873 - 95 -- 0.938 - 93
- %80 train %20 test = 0.939 - 63

5.2 30% test 70% train

In [65]:

```
#Seleccionamos nuestros datos en X,y
X1 = datanormal.drop("Population_Driver_licence(%)",axis=1) # Variables independientes
y1 = datanormal["Population_Driver_licence(%)"] # Variable dependiente
#Entrenamos modelo y realizamos partición de datos para testear y entrenar

X_train1,X_test1,y_train1,y_test1 = train_test_split(X1,y1,test_size = 0.3, random_sta

#Crando modelo de regresión lineal
modelo1 = LinearRegression()
modelo1.fit(X_train1,y_train1)

#Predicción de los datos de testeo
y_pred1 = modelo1.predict (X_test1)

#Calculamos metricas
mse1 = mean_squared_error(y_test1,y_pred1)
rmse1 = np.sqrt(mse1)
r21 = round(r2_score(y_test1,y_pred1),3)

print("Error cuadratico medio (MSE) : ",mse1)
print("Error Cuadratico Medio (RMSE) : ",rmse1)
print("Coeficiente de Determminacion (R2): ",r21)
```

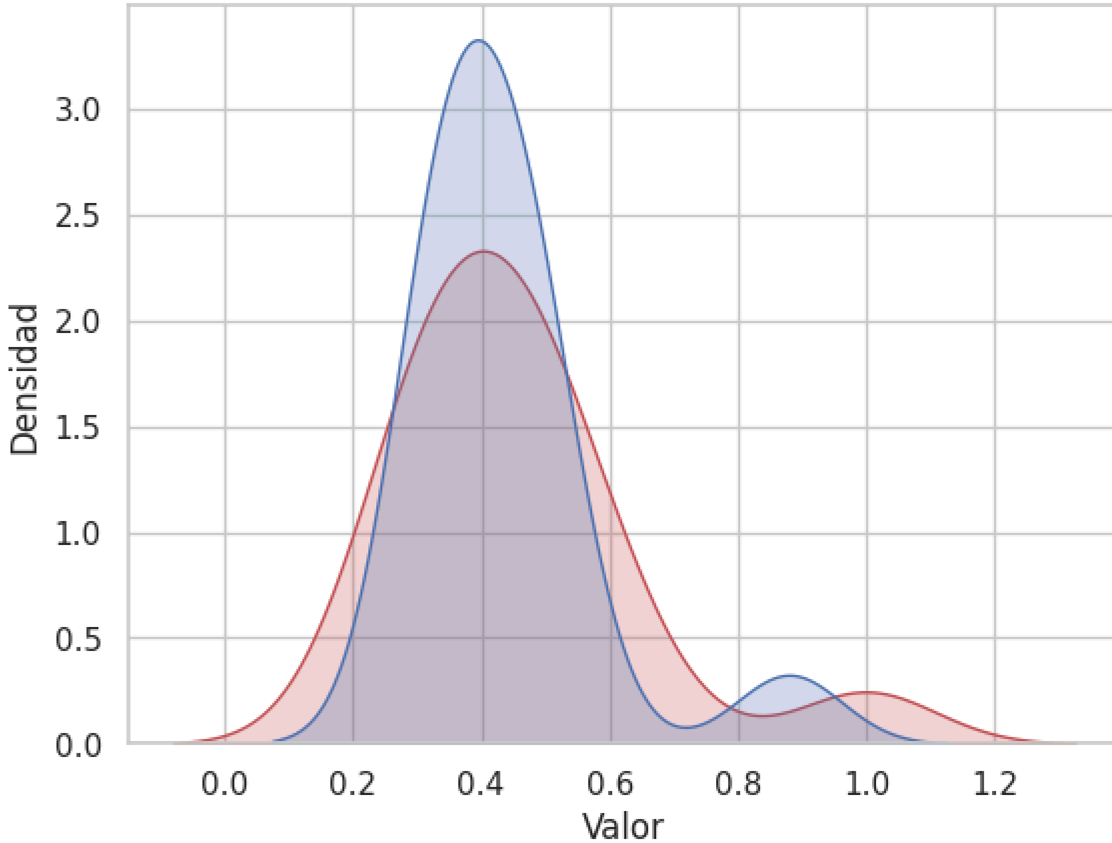
```
Error cuadratico medio (MSE) :  0.006642343889957654
Error Cuadratico Medio (RMSE) :  0.08150057600997464
Coeficiente de Determminacion (R2):  0.795
```

In [66]:

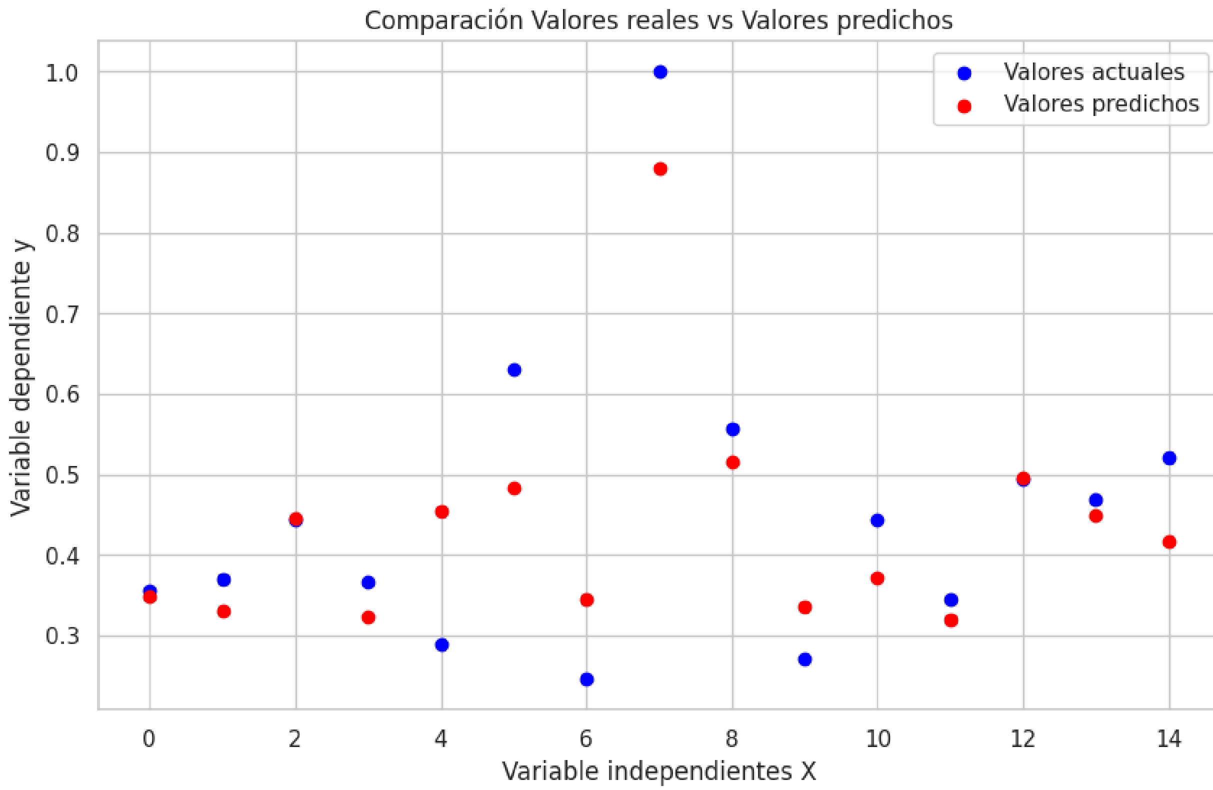
```
import seaborn as sns
ax1 = sns.kdeplot(y_test1,color="r",label="Valores actuales",fill = True)
sns.kdeplot(y_pred1,color="b", label="Valores predecidos", fill = True)

plt.xlabel("Valor")
plt.ylabel("Densidad")
```

Out[66]:

```
Text(0, 0.5, 'Densidad')
```



In [67]:
```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test1)), y_test1, color='blue', label='Valores actuales')
plt.scatter(range(len(y_pred1)), y_pred1, color='red', label='Valores predichos')
plt.xlabel('Variable independientes X')
plt.ylabel('Variable dependiente y')
plt.title('Comparación Valores reales vs Valores predichos')
plt.legend()
plt.show()
```



In [68]:
```
X1 =X_test1
X1= sm.add_constant(X1)
y1=y_test1

modelo_ols1 = sm.OLS(y1,X1).fit(cov_type="HC3")
print(modelo_ols1.summary())
```

```
                                  OLS Regression Results
=====================================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                       0.
873
Model:                                      OLS   Adj. R-squared:                  0.
822
Method:                           Least Squares   F-statistic:                     3
3.03
Date:                          Sat, 04 May 2024   Prob (F-statistic):           9.74e
-06
Time:                                  16:46:03   Log-Likelihood:                 19.
895
No. Observations:                            15   AIC:                            -2
9.79
Df Residuals:                                10   BIC:                            -2
6.25
Df Model:                                     4
Covariance Type:                            HC3
=====================================================================================
=
                         coef    std err          z      P>|z|      [0.025      0.97
5]
-------------------------------------------------------------------------------------
-
const                  0.0336      0.249      0.135      0.893      -0.455       0.52
2
Petrol_tax            -0.1378      0.273     -0.505      0.614      -0.673       0.39
7
Average_income         0.4385      0.140      3.121      0.002       0.163       0.71
4
Paved_Highways        -0.2808      0.194     -1.445      0.148      -0.662       0.10
0
Petrol_Consumption     0.9358      0.107      8.707      0.000       0.725       1.14
6
==============================================================================
Omnibus:                       10.264   Durbin-Watson:                   2.470
Prob(Omnibus):                  0.006   Jarque-Bera (JB):                6.437
Skew:                          -1.333   Prob(JB):                       0.0400
Kurtosis:                       4.786   Cond. No.                         21.9
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

5.3 20% test 80% train

In [69]:

```
#Seleccionamos nuestros datos en X,y
X2 = datanormal.drop("Population_Driver_licence(%)",axis=1) # Variables independientes
y2 = datanormal["Population_Driver_licence(%)"] # Variable dependiente
#Entrenamos modelo y realizamos partición de datos para testear y entrenar

X_train2,X_test2,y_train2,y_test2 = train_test_split(X2,y2,test_size = 0.2, random_sta

#Crando modelo de regresión lineal
modelo2 = LinearRegression()
modelo2.fit(X_train2,y_train2)
```

```
#Predicción de los datos de testeo
y_pred2 = modelo2.predict (X_test2)

#Calculamos metricas
mse2 = mean_squared_error(y_test2,y_pred2)
rmse2 = np.sqrt(mse2)
r22 = round(r2_score(y_test2,y_pred2),3)

print("Error cuadratico medio (MSE) : ",mse2)
print("Error Cuadratico Medio (RMSE) : ",rmse2)
print("Coeficiente de Determminacion (R2): ",r22)
```

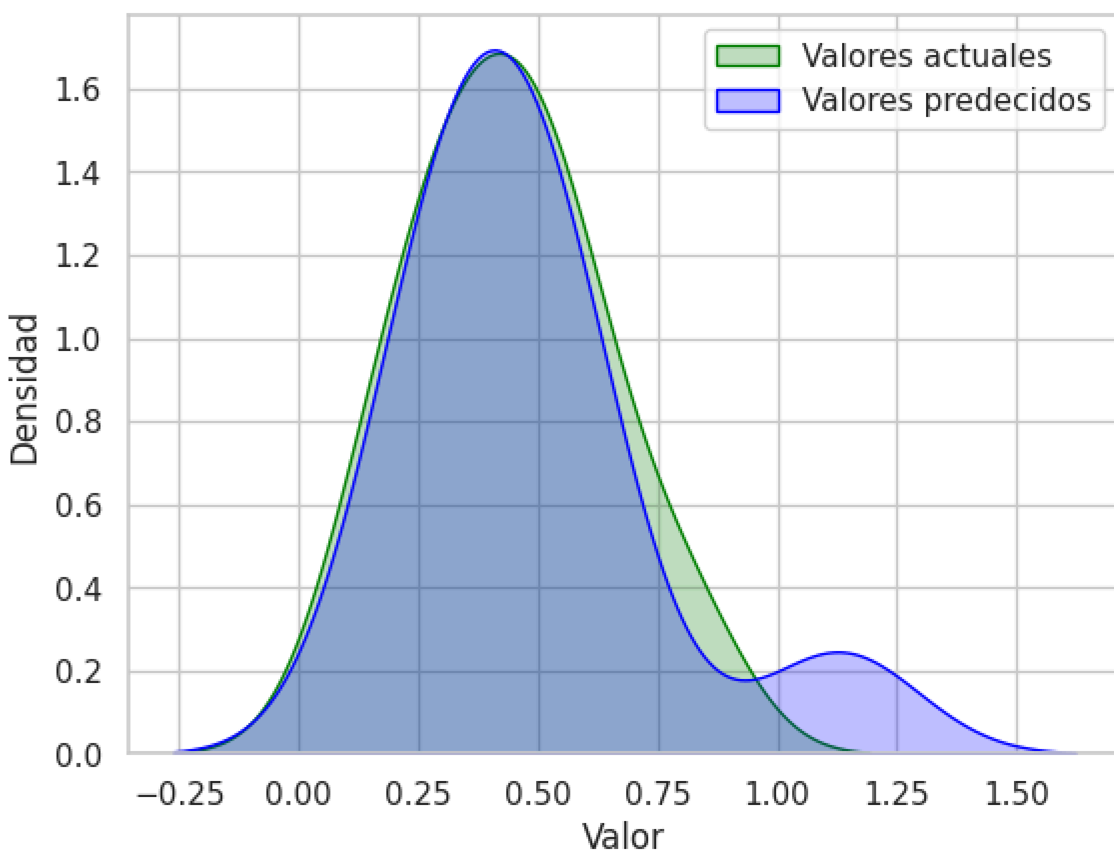
```
Error cuadratico medio (MSE) :  0.01889938373952545
Error Cuadratico Medio (RMSE) :  0.13747502951272805
Coeficiente de Determminacion (R2):  0.483
```

In [70]:

```
import seaborn as sns

sns.set(style = "whitegrid")
ax2 = sns.kdeplot(y_test2,color="green",label="Valores actuales",fill = True)
sns.kdeplot(y_pred2,color="blue", label="Valores predecidos", fill = True)

plt.legend()
plt.xlabel("Valor")
plt.ylabel("Densidad")
```

Out[70]:

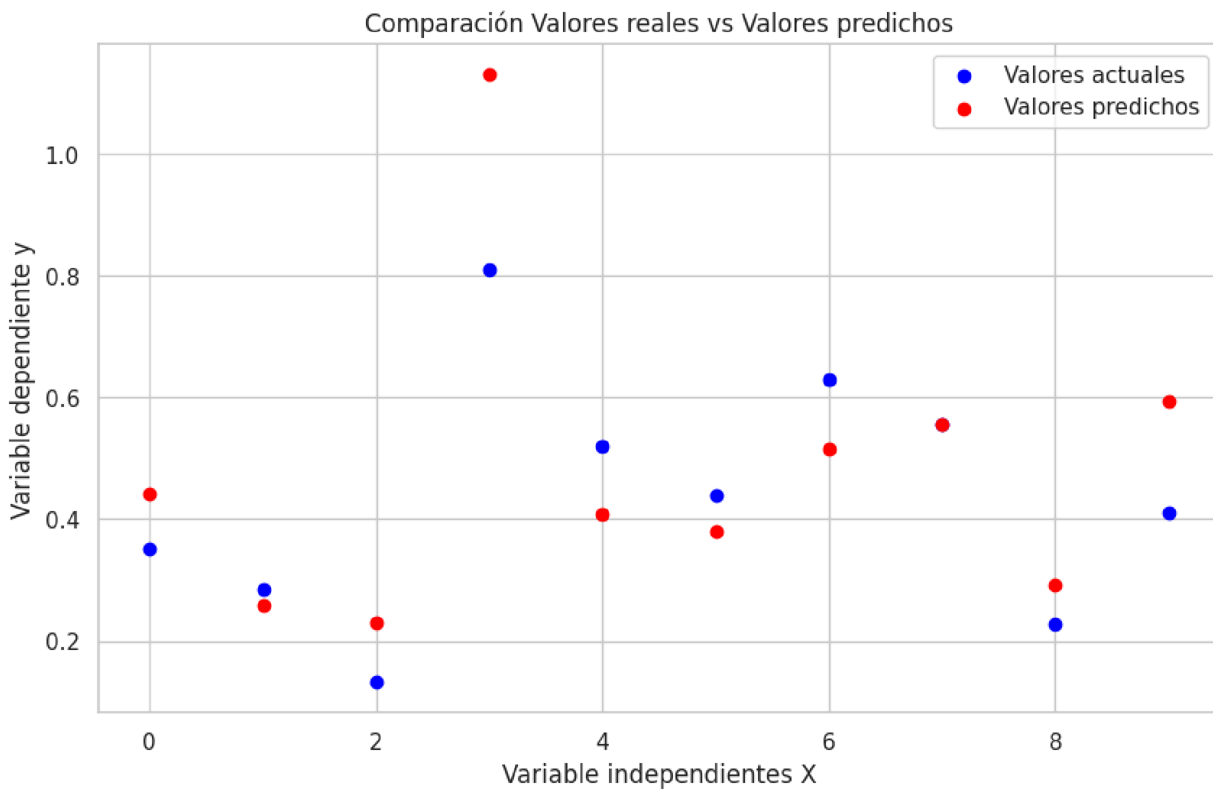
```
Text(0, 0.5, 'Densidad')
```



In [71]:

```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test2)), y_test2, color='blue', label='Valores actuales')
plt.scatter(range(len(y_pred2)), y_pred2, color='red', label='Valores predichos')
plt.xlabel('Variable independientes X')
```

```
plt.ylabel('Variable dependiente y')
plt.title('Comparación Valores reales vs Valores predichos')
plt.legend()
plt.show()
```



In [72]:
```
X2 =X_test2
X2 = sm.add_constant(X2)
y2 = y_test2

modelo_ols2 = sm.OLS(y2,X2).fit(cov_type="HC3")
print(modelo_ols2.summary())
```

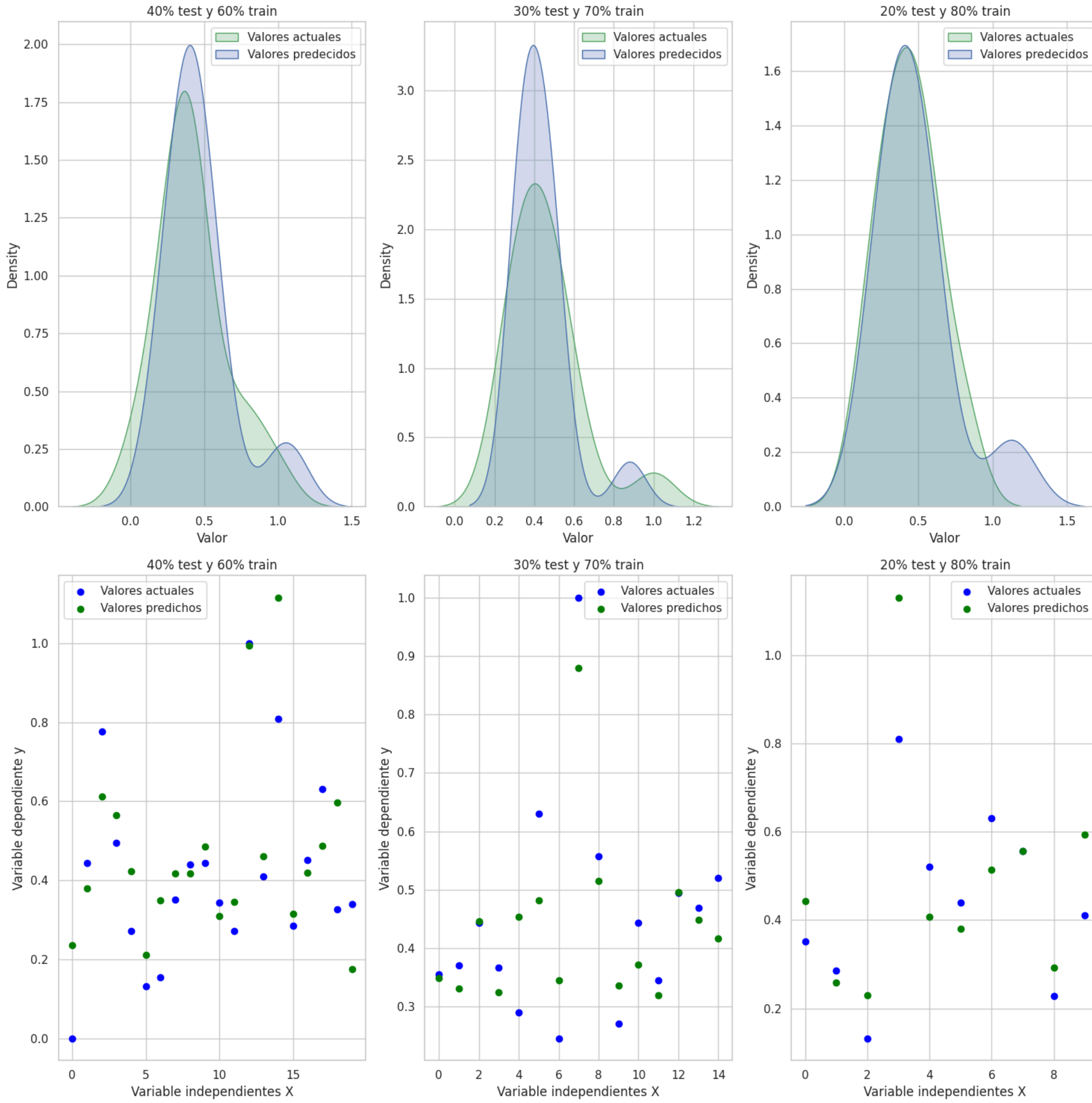
```
                                 OLS Regression Results
====================================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                       0.
939
Model:                                      OLS   Adj. R-squared:                  0.
890
Method:                           Least Squares   F-statistic:                     8.
241
Date:                          Sat, 04 May 2024   Prob (F-statistic):             0.0
200
Time:                                  16:46:04   Log-Likelihood:                 16.
324
No. Observations:                            10   AIC:                            -2
2.65
Df Residuals:                                 5   BIC:                            -2
1.14
Df Model:                                     4
Covariance Type:                            HC3
====================================================================================
=
                         coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------------
-
const                  0.2832      0.295      0.958      0.338      -0.296       0.86
2
Petrol_tax            -0.3419      0.361     -0.946      0.344      -1.050       0.36
7
Average_income         0.6013      0.137      4.393      0.000       0.333       0.87
0
Paved_Highways        -0.4704      0.309     -1.523      0.128      -1.076       0.13
5
Petrol_Consumption     0.4344      0.262      1.655      0.098      -0.080       0.94
9
==============================================================================
Omnibus:                        0.148   Durbin-Watson:                   1.746
Prob(Omnibus):                  0.929   Jarque-Bera (JB):                0.350
Skew:                          -0.024   Prob(JB):                        0.839
Kurtosis:                       2.085   Cond. No.                         21.8
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

6. Escogiendo mejor modelo

In [73]:

```
# Crea figura y multiples subplots
fig, axes = plt.subplots(2, 3, figsize=(15, 15))

# Graficar para 40% test y 60% train
#Diagrama de densidad
sns.kdeplot(y_test, color="g", label="Valores actuales", fill=True, ax=axes[0, 0])
sns.kdeplot(y_pred, color="b", label="Valores predecidos", fill=True, ax=axes[0, 0])
axes[0, 0].set_xlabel("Valor") # Label eje X
axes[0, 0].set_title("40% test y 60% train") # Titulo
axes[0, 0].legend()
#Diagrama de puntos
```

```
axes[1, 0].scatter(range(len(y_test)), y_test, color='blue', label='Valores actuales')
axes[1, 0].scatter(range(len(y_pred)), y_pred, color='green', label='Valores predichos
axes[1,0].set_xlabel('Variable independientes X')
axes[1,0].set_ylabel('Variable dependiente y')
axes[1, 0].set_title("40% test y 60% train")
axes[1, 0].legend()
#--------------------------------------------------------------------------------
# Graficar para 30% test y 70% train
#Diagrama de densidad
sns.kdeplot(y_test1, color="g", label="Valores actuales", fill=True, ax=axes[0, 1])
sns.kdeplot(y_pred1, color="b", label="Valores predecidos", fill=True, ax=axes[0, 1])
axes[0, 1].set_xlabel("Valor")
axes[0, 1].set_title("30% test y 70% train")
axes[0, 1].legend()
# Diagrama de puntos
axes[1, 1].scatter(range(len(y_test1)), y_test1, color='blue', label='Valores actuales
axes[1, 1].scatter(range(len(y_pred1)), y_pred1, color='green', label='Valores predich
axes[1,1].set_xlabel('Variable independientes X')
axes[1,1].set_ylabel('Variable dependiente y')
axes[1, 1].set_title("30% test y 70% train")
axes[1, 1].legend()
#--------------------------------------------------------------------------------
# Graficar para 20% test y 80% train
#Diagrama de densidad
sns.kdeplot(y_test2, color="g", label="Valores actuales", fill=True, ax=axes[0, 2])
sns.kdeplot(y_pred2, color="b", label="Valores predecidos", fill=True, ax=axes[0, 2])
axes[0, 2].set_xlabel("Valor")
axes[0, 2].set_title("20% test y 80% train")
axes[0, 2].legend()
# Diagrama de puntos
axes[1, 2].scatter(range(len(y_test2)), y_test2, color='blue', label='Valores actuales
axes[1, 2].scatter(range(len(y_pred2)), y_pred2, color='green', label='Valores predich
axes[1,2].set_xlabel('Variable independientes X')
axes[1,2].set_ylabel('Variable dependiente y')
axes[1, 2].set_title("20% test y 80% train")
axes[1, 2].legend()


# Poder visualizar las graficas
plt.tight_layout()
plt.show()
```



In [74]:

```python
# Modelo 40% test y 60% train
X =X_test
X = sm.add_constant(X)
y = y_test

modelo_ols = sm.OLS(y,X).fit(cov_type="HC3")
print(modelo_ols.summary())
```

```
                                 OLS Regression Results                                 
========================================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                         0.
811
Model:                                      OLS   Adj. R-squared:                    0.
760
Method:                           Least Squares   F-statistic:                       7.
619
Date:                          Sat, 04 May 2024   Prob (F-statistic):             0.00
147
Time:                                  16:46:06   Log-Likelihood:                   17.
400
No. Observations:                            20   AIC:                              -2
4.80
Df Residuals:                                15   BIC:                              -1
9.82
Df Model:                                     4                                         
Covariance Type:                            HC3                                         
========================================================================================
=
                         coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------------------
-
const                  0.1248      0.242      0.516      0.606      -0.349       0.59
9
Petrol_tax            -0.2289      0.202     -1.136      0.256      -0.624       0.16
6
Average_income         0.5271      0.195      2.705      0.007       0.145       0.90
9
Paved_Highways        -0.4598      0.142     -3.243      0.001      -0.738      -0.18
2
Petrol_Consumption     0.7617      0.289      2.637      0.008       0.196       1.32
8
==============================================================================
Omnibus:                        2.190   Durbin-Watson:                   1.807
Prob(Omnibus):                  0.334   Jarque-Bera (JB):                1.018
Skew:                          -0.537   Prob(JB):                        0.601
Kurtosis:                       3.263   Cond. No.                         17.5
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

In [75]:

```python
# Modelo 30% test y 70% train
X1 =X_test1
X1 = sm.add_constant(X1)
y1 = y_test1

modelo_ols1 = sm.OLS(y1,X1).fit(cov_type="HC3")
print(modelo_ols1.summary())
```

```
                            OLS Regression Results
==============================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                       0.
873
Model:                                      OLS   Adj. R-squared:                  0.
822
Method:                           Least Squares   F-statistic:                     3
3.03
Date:                          Sat, 04 May 2024   Prob (F-statistic):           9.74e
-06
Time:                                  16:46:06   Log-Likelihood:                 19.
895
No. Observations:                            15   AIC:                            -2
9.79
Df Residuals:                                10   BIC:                            -2
6.25
Df Model:                                     4
Covariance Type:                            HC3
==============================================================================
=
                        coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
const                 0.0336      0.249      0.135      0.893      -0.455       0.52
2
Petrol_tax           -0.1378      0.273     -0.505      0.614      -0.673       0.39
7
Average_income        0.4385      0.140      3.121      0.002       0.163       0.71
4
Paved_Highways       -0.2808      0.194     -1.445      0.148      -0.662       0.10
0
Petrol_Consumption    0.9358      0.107      8.707      0.000       0.725       1.14
6
==============================================================================
Omnibus:                       10.264   Durbin-Watson:                   2.470
Prob(Omnibus):                  0.006   Jarque-Bera (JB):                6.437
Skew:                          -1.333   Prob(JB):                       0.0400
Kurtosis:                       4.786   Cond. No.                         21.9
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

In [76]:

```python
# Modelo 20% test y 80% train
X2 =X_test2
X2 = sm.add_constant(X2)
y2 = y_test2

modelo_ols2 = sm.OLS(y2,X2).fit(cov_type="HC3")
print(modelo_ols2.summary())
```

```
                             OLS Regression Results
=====================================================================================
===
Dep. Variable:     Population_Driver_licence(%)   R-squared:                       0.
939
Model:                                      OLS   Adj. R-squared:                  0.
890
Method:                           Least Squares   F-statistic:                     8.
241
Date:                          Sat, 04 May 2024   Prob (F-statistic):             0.0
200
Time:                                  16:46:06   Log-Likelihood:                 16.
324
No. Observations:                            10   AIC:                            -2
2.65
Df Residuals:                                 5   BIC:                            -2
1.14
Df Model:                                     4
Covariance Type:                            HC3
=====================================================================================
=
                         coef    std err          z      P>|z|      [0.025      0.97
5]
-------------------------------------------------------------------------------------
-
const                  0.2832      0.295      0.958      0.338      -0.296       0.86
2
Petrol_tax            -0.3419      0.361     -0.946      0.344      -1.050       0.36
7
Average_income         0.6013      0.137      4.393      0.000       0.333       0.87
0
Paved_Highways        -0.4704      0.309     -1.523      0.128      -1.076       0.13
5
Petrol_Consumption     0.4344      0.262      1.655      0.098      -0.080       0.94
9
==============================================================================
Omnibus:                        0.148   Durbin-Watson:                   1.746
Prob(Omnibus):                  0.929   Jarque-Bera (JB):                0.350
Skew:                          -0.024   Prob(JB):                        0.839
Kurtosis:                       2.085   Cond. No.                         21.8
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

Se escogio el modelo de 20% test y 80% train, porque al aplicarle el modelo OLS se logro una alta mejoría en su eficiencia.

Ademas este es un modelo muy estable y bueno para predecir **el porcentaje de población con licencia de conducir** con base en las variables independientes de X, mostrando que la mejor variable que la ayuda a predecir es el **Ingreso promedio** y **el consumo de gasolina**. También el modelo es estable a futuro teniendo un R2 muy cercano al R2 ajustado con una diferencia de 0.049 puntos porcentuales.

7. Conclusiones

1. Se realizo la exploración de la data en la cual se encontró que no existían valores Nulos (NaN), posterior a eso se ejecuta la normalización de los datos con la función MinMaxScaler porque las desviasiones estándar de las variables estaban separadas unas de otras. Luego se realizo el test de normalidad de shapirowik donde solo se evidencia la presencia de 2 variables que son normales las cuales eran **Average_income** y **Population_Driver_licence(%)**, eso quiere decir que son buenas opciones para predecir, donde se opto por la variable **Population_Driver_licence(%)**.

2. Aplicamos una regresión lineal multiple para encontrar que relación tienen las variables independiente con nuestra variable dependiente **Population_Driver_licence(%)**.

Para hacer la regresión lineal multiple se realiaron los siguientes pasos:

- Primero se realizo la revisión de homoscedasticidad con el test de breush pagan donde se pudo evidenciar que las todas las variables tienen homoscedasticidad de 0.17, lo que es bastante bueno.

- Segundo se reviso la multicolinealidad utilizando el modelo de VIF, donde se pudo identificar que ninguna variable presenta un caso alto de multicolinealidad porque casi todas estaban en un rango entre (1-2),lo que representa un varianza de inflación buena.

Como el VIF y la homocedasticidad son buenas,se realizo el temas de las predicciones con 3 particiones diferentes de entrenamiento y testeo y se aplicarón los modelos de regresión lineal de la librería de sckity learn y el modelo de OLS de la librería statsmodel.

1. La primer partición fue de un 40% test y 60% train; donde se entreno el modelo de regresión lineal y nos dio una eficiencia de 0.63 y con MSE y RMSE cercanos a 0. Ya para el modelo OLS se evidencia una mejoría en la eficiencia con un 0.81.

2. En la segunda partición fue de un 30% test y 70% train; donde se entreno el modelo de regresión lineal con una eficiencia de 0.79 y con MSE y RMSE cercanos a 0. Ya para el modelo OLS se evidencia una mejoría en la eficiencia con un 0.87.

3. En la tercera partición fue de un 20% test y 80% train; donde se entreno el modelo de regresión lineal con una eficiencia de 0.48 y con MSE y RMSE cercanos a 0. Ya para el modelo OLS se evidencia una mejoría en la eficiencia con un 0.93.

Al comparar la eficiencia de las 3 particiones de porcecntajes diferentes, se escogio la tercera porque al aplicarle el modelo OLS se logro una alta mejoría en su eficiencia.

Ademas este es un modelo muy estable y bueno para predecir **el porcentaje de población con licencia de conducir** con base en las variables independientes de X, mostrando que la mejor variable que la ayuda a predecir es el **Ingreso promedio** y **el consumo de gasolina**. También el modelo es estable a futuro teniendo un R2 muy cercano al R2 ajustado con una diferencia de 0.049 puntos porcentuales.

Este modelo hace referencia a que de 100 datos de el porcentaje de población con licencia de conducir es posible predecir un 93% de manera eficiente.

8. DESNORMALIZANDO DATOS

Para el que de mejor presicion

In [77]:
```
#Desnormalizando y_test y y_pred
y_test_desnormalizado = y_test2 * (1 - 0) + 0 # rangos de 0-1 por MinMaxScaler
y_pred_desnormalizado = y_pred2 * (1 - 0) + 0

#Creando un DataFrame para guardar los datos desnormalizados de y_test2 y y_pred2
df_desnormalizado = pd.DataFrame({'y_test_desnormalizado': y_test_desnormalizado, 'y_p

#Visualizando el dataframe
print(df_desnormalizado)
```

```
    y_test_desnormalizado  y_pred_desnormalizado
33               0.351648               0.442165
8                0.285714               0.258954
34               0.131868               0.229750
39               0.809524               1.130524
47               0.520147               0.407992
45               0.439560               0.380601
46               0.630037               0.514569
42               0.556777               0.555617
31               0.227106               0.292391
41               0.410256               0.594136
```