MAESTRÍA EN CIENCIA DE DATOS

FUNDAMENTOS CIENCIA DE DATOS

PROYECTO #2

TEMA:

"Modelo de Predicción del Consumo de Diésel en Unidades de Transporte Publico"**.**

FEATURING ENGINEERING

<u>Creación de Nuevas Variables</u>

Variables Temporales: Estas variables permiten identificar patrones de consumo que pueden estar relacionados con el calendario. Por ejemplo, ciertos días de la semana o meses podrían tener más demanda de combustible.

```
c_diesel['DIA_SEMANA'] = c_diesel['FECHA'].dt.dayofweek
c_diesel['MES'] = c_diesel['FECHA'].dt.month
c_diesel['DIA'] = c_diesel['FECHA'].dt.day
c_diesel['ANIO'] = c_diesel['FECHA'].dt.year
```

Corregimos los valores de "GALONES" que utilicen separador de decimal.

```
c_diesel['GALONES'] = c_diesel['GALONES'].astype(float)
✓ 0.0s
```

Reemplazar los ceros en kilometraje para evitar división por cero, lo que ocasionaría una indeterminación.

```
c_diesel = c_diesel[c_diesel['KILOMETRAJE'] != 0]
✓ 0.0s
```

Variables Derivadas: Se crea nueva variable "GALONES_POR_KM", obtenida dividiendo los galones consumidos por el kilometraje correspondiente:

```
c_diesel['GALONES_POR_KM'] = c_diesel['GALONES'] / c_diesel['KILOMETRAJE']
✓ 0.0s
```

Esta variable representa el rendimiento del consumo de combustible, lo cual puede ser más representativo para el modelo que únicamente analizar "GALONES" y "KILOMETRAJE".

Codificación de Variables Categóricas: Se aplicó LabelEncoder a las variables categóricas. Los modelos de Machine Learning no pueden trabajar directamente con strings, por lo que es necesario codificar estas variables.

```
le_vehiculo = LabelEncoder()
le_flota = LabelEncoder()
c_diesel['VEHICULO_ENC'] = le_vehiculo.fit_transform(c_diesel['VEHICULO'])
c_diesel['FLOTA_ENC'] = le_flota.fit_transform(c_diesel['FLOTA'])
c_diesel.head()
```

✓ 0.0s

Selección de Variables

Se analizan las variables con poca variabilidad. Lo que indica que variables con desviación estándar muy baja pueden eliminarse.

```
variabilidad = c_diesel.std(numeric_only=True).sort_values()
print("Variables con menor variabilidad:\n", variabilidad)
```

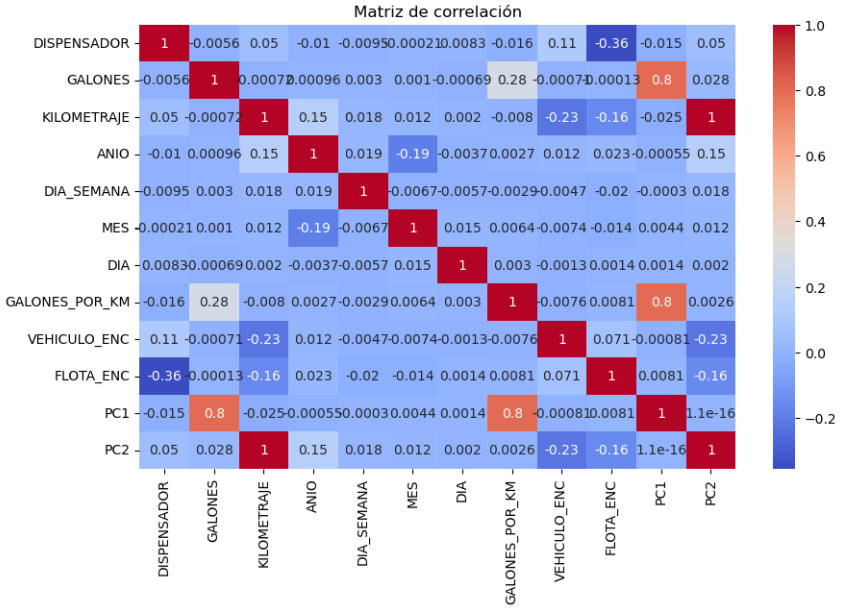
✓ 0.0s

```
Variables con menor variabilidad:
 ANIO              1.070067e+00
DISPENSADOR        1.204456e+00
DIA_SEMANA         1.911405e+00
FLOTA_ENC          1.911857e+00
MES                3.411460e+00
DIA                8.752879e+00
VEHICULO_ENC       8.715255e+01
GALONES_POR_KM     1.415496e+03
KILOMETRAJE        3.144600e+05
GALONES            1.360346e+08
dtype: float64
```

Se elabora una matriz de correlación, en el caso de que dos variables tienen correlación > 0.95, se puede conservar solo una para evitar redundancia.

```
corr_matrix = c_diesel.corr(numeric_only=True)
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Matriz de correlación")
plt.show()
```

✓ 0.5s



Reducimos la dimensionalidad para que sea más simple detectar combinaciones lineales útiles entre variables correlacionadas, para ello realizamos la depuración del dataset. En caso de existir valores infinitos, estos serán reemplazados por NaN.

```
c_diesel.replace([np.inf, -np.inf], np.nan, inplace=True)
c_diesel.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
Index: 192946 entries, 0 to 193390
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   FECHA           192946 non-null  datetime64[ns]
 1   DISPENSADOR     192946 non-null  int64
 2   HORA            192943 non-null  object
 3   VEHICULO        192946 non-null  object
 4   FLOTA           192946 non-null  object
 5   GALONES         192943 non-null  float64
 6   KILOMETRAJE     192946 non-null  float64
 7   ANIO            192946 non-null  int32
 8   DIA_SEMANA      192946 non-null  int32
 9   MES             192946 non-null  int32
 10  DIA             192946 non-null  int32
 11  GALONES_POR_KM  192943 non-null  float64
 12  VEHICULO_ENC    192946 non-null  int32
 13  FLOTA_ENC       192946 non-null  int32
dtypes: datetime64[ns](1), float64(3), int32(6), int64(1), object(3)
memory usage: 17.7+ MB
```

Se eliminan posibles valores nulos.

```
features = ['GALONES', 'KILOMETRAJE', 'GALONES_POR_KM']
c_diesel.dropna(subset=features, inplace=True)
c_diesel.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 192943 entries, 0 to 193390
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   FECHA           192943 non-null  datetime64[ns]
 1   DISPENSADOR     192943 non-null  int64
 2   HORA            192940 non-null  object
 3   VEHICULO        192943 non-null  object
 4   FLOTA           192943 non-null  object
 5   GALONES         192943 non-null  float64
 6   KILOMETRAJE     192943 non-null  float64
 7   ANIO            192943 non-null  int32
 8   DIA_SEMANA      192943 non-null  int32
 9   MES             192943 non-null  int32
 10  DIA             192943 non-null  int32
 11  GALONES_POR_KM  192943 non-null  float64
 12  VEHICULO_ENC    192943 non-null  int32
 13  FLOTA_ENC       192943 non-null  int32
dtypes: datetime64[ns](1), float64(3), int32(6), int64(1), object(3)
memory usage: 17.7+ MB
```

Transformamos los datos numéricos para que todas las variables estén en la misma escala o rango de valores para equilibrar el análisis de los datos.

```
X_scaled = StandardScaler().fit_transform(c_diesel[features])
```

✓ 0.0s

Aplicamos PCA ya que este reducir dimensionalidad y detectar combinaciones lineales útiles entre variables correlacionadas.

```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)
```

✓ 0.0s

Añadimos al dataset:

```
c_diesel['PC1'] = principal_components[:, 0]
c_diesel['PC2'] = principal_components[:, 1]
c_diesel.info()
```

✓ 0.0s

Donde podemos observar todas las variables nuevas creadas:

```
<class 'pandas.core.frame.DataFrame'>
Index: 192943 entries, 0 to 193390
Data columns (total 16 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   FECHA           192943 non-null  datetime64[ns]
 1   DISPENSADOR     192943 non-null  int64
 2   HORA            192940 non-null  object
 3   VEHICULO        192943 non-null  object
 4   FLOTA           192943 non-null  object
 5   GALONES         192943 non-null  float64
 6   KILOMETRAJE     192943 non-null  float64
 7   ANIO            192943 non-null  int32
 8   DIA_SEMANA      192943 non-null  int32
 9   MES             192943 non-null  int32
 10  DIA             192943 non-null  int32
 11  GALONES_POR_KM  192943 non-null  float64
 12  VEHICULO_ENC    192943 non-null  int32
 13  FLOTA_ENC       192943 non-null  int32
 14  PC1             192943 non-null  float64
 15  PC2             192943 non-null  float64
dtypes: datetime64[ns](1), float64(5), int32(6), int64(1), object(3)
memory usage: 20.6+ MB
```

Justificación

Variables Creadas:

- **DIA_SEMANA, MES, DIA, ANIO:** Aportan información temporal relevante para entender patrones de consumo.
- **GALONES_POR_KM: mide eficiencia del consumo.**
- **VEHICULO_ENC, FLOTA_ENC:** Transformaciones necesarias para que los modelos entiendan variables categóricas.
- **PC1, PC2:** Componentes principales que resumen información si se decide reducir dimensionalidad.

MODELADO (MODELING)

Definimos las características (X) y las variables objetivo (Y).

```
features = ['KILOMETRAJE', 'GALONES_POR_KM', 'ANIO']
target = 'GALONES'

X = f_diesel[features]
y = f_diesel[target]
```
✓ 0.0s

Se realiza un Escalado.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```
✓ 0.0s

Dividimos en entrenamiento y prueba:

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)
```
✓ 0.0s

Creamos u diccionario con los modelos que vamos a aplicar:

```
modelos = {
    'LinearRegression': LinearRegression(),
    'DecisionTreeRegressor': DecisionTreeRegressor(random_state=100),
    'RandomForestRegressor': RandomForestRegressor(random_state=100),
    'GradientBoostingRegressor': GradientBoostingRegressor(random_state=100),
}
```
✓ 0.0s

Evaluamos los modelos seleccionados:

```
resultados = {}

for nombre, modelo in modelos.items():
    modelo.fit(X_train, y_train)
    y_pred = modelo.predict(X_test)

    resultados[nombre] = {
        'MAE': mean_absolute_error(y_test, y_pred),
        'RMSE': mean_squared_error(y_test, y_pred, squared=False),
        'R2': r2_score(y_test, y_pred)
    }
```
✓ 1m 15.4s

Obteniendo los siguientes resultados:

```
df_resultados = pd.DataFrame(resultados).T.sort_values(by='RMSE')
print("Resultados del rendimiento de los modelos:\n")
print(df_resultados)
```
✓ 0.0s

```
Resultados del rendimiento de los modelos:

                                MAE          RMSE        R2
RandomForestRegressor      50627.788493  8.831130e+06  0.963653
DecisionTreeRegressor      60404.052506  1.272420e+07  0.924543
GradientBoostingRegressor  110749.452584  1.275680e+07  0.924155
LinearRegression           545643.078395  3.896687e+07  0.292327
```

Conclusiones:

- Random Forest fue el modelo con mejor rendimiento (0.96), indicando que captura muy bien las relaciones entre variables.
- Linear Regression tuvo un bajo rendimiento (0.29), lo que sugiere que la relación entre las variables no es puramente lineal.
- Los métodos basados en árboles (DecisionTree y GradientBoosting) ofrecieron buenos resultados, aunque inferiores al RandomForest.

Predicción Futura y Visualización

- Se utilizó el mejor modelo (RandomForestRegressor) para realizar predicciones de consumo en fechas futuras usando datos promedio.
- Estas predicciones se graficaron junto con el histórico, mostrando la evolución del KILOMETRAJE estimado.
- Justificación: La visualización permite entender cómo se proyecta el consumo en el futuro. Esto es útil para la planificación operativa y logística.

Estructura del Repositorio de GitHub

```
Proyeccion_Consumo_Diesel/
│
├─── data/                                  # Datos en crudo y limpios
│    ├─── Consumo_Diesel.csv                # Dataset original
│    ├─── Consumo_Diesel_Limpio.csv         # Dataset modificado en Feature Engineering
│    ├─── Consumo_Diesel_Model.csv          # Dataset final para modelado
│    └─── Consumo_Diesel_mLimpio.csv        # Dataset modificado en Wrangling
│
├─── notebooks/                             # Notebooks(scripts)
│    ├─── 1_EDA_Diesel.ipynb
│    ├─── 2_Wrangling_Diesel.ipynb
│    ├─── 3_Feature_Engineering_Diesel.ipynb
│    └─── Consumo_Diesel.ipynb
│
├─── reports/                               # Reportes o presentaciones
│    ├─── Reporte_1.pdf
│    └─── Reporte_2.pdf
│
├─── .gitignore              # Ignorar archivos innecesarios (ej. *.pyc, .ipynb_checkpoints)
└─── README.md               # Descripción del proyecto
```