



Analítico asignación de viaje

Análisis de ciencia de datos TC2004B.201



Profesores:
Felipe Castillo Rendón
María de los Ángeles Constantino González

Equipo 5

Kevin Antonio González Díaz
A01338316



- Gráficas pruebas de dependencia
- Balanceo de datos
- Optimización de hiperparámetros
- Matrices de confusión y mapas de calor de modelos de aprendizaje
- Elección modelo de aprendizaje
- Lectura de archivo y generación de predicciones.

Alexis Daniel Leyva Yáñez
A01770308



- Limpieza de los datos: Manejo de los valores faltantes
- Ajuste de los datos outliers (atípicos)
- Transformación de datos categóricos a numéricos
- Discretización de los datos
- Reducción de ruido
- Evaluación de modelos de aprendizaje automático

Luis Maximiliano López Ramírez
A00833321



- Integración de las bases de datos
- Limpieza de los datos
- Gráficas de variables cuantitativas
- Transformación de datos categóricos a numéricos
- Codificación de los modelos de aprendizaje supervisado de clasificación

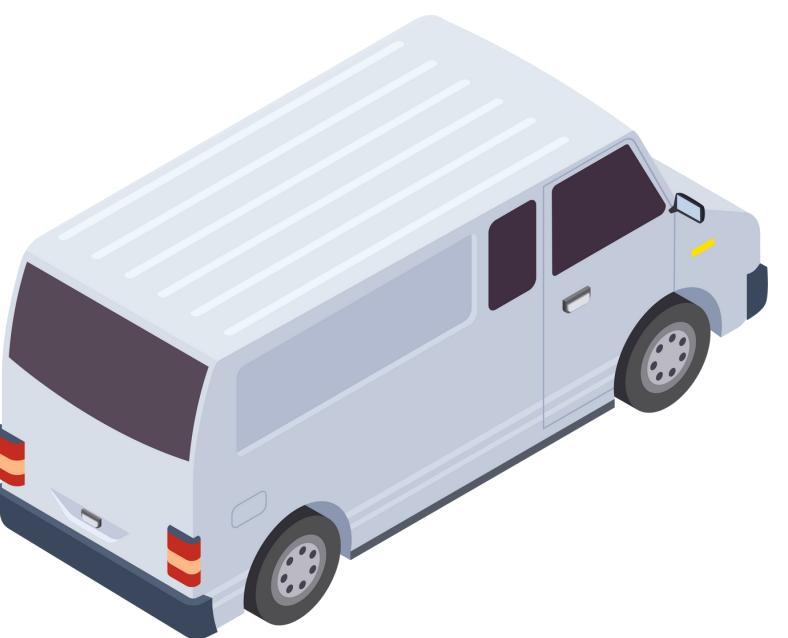
Adrián Pineda Sánchez
A00834710



- Limpieza general de la base de datos
- División modelo de datos en entrenamiento y prueba
- Descripción e Implementación de los modelos de aprendizaje automático.
- Gráficas de evaluación de métricas de los modelos.
- Selección de modelo base para optimizar hiperparámetros

Contenido

- 1 Objetivo principal
- 2 Fuentes de datos
- 3 Exploración de Datos
- 4 Generación y evaluación de modelos de aprendizaje
- 5 Selección y despliegue del modelo
- 6 Resumen, recomendaciones y conclusiones

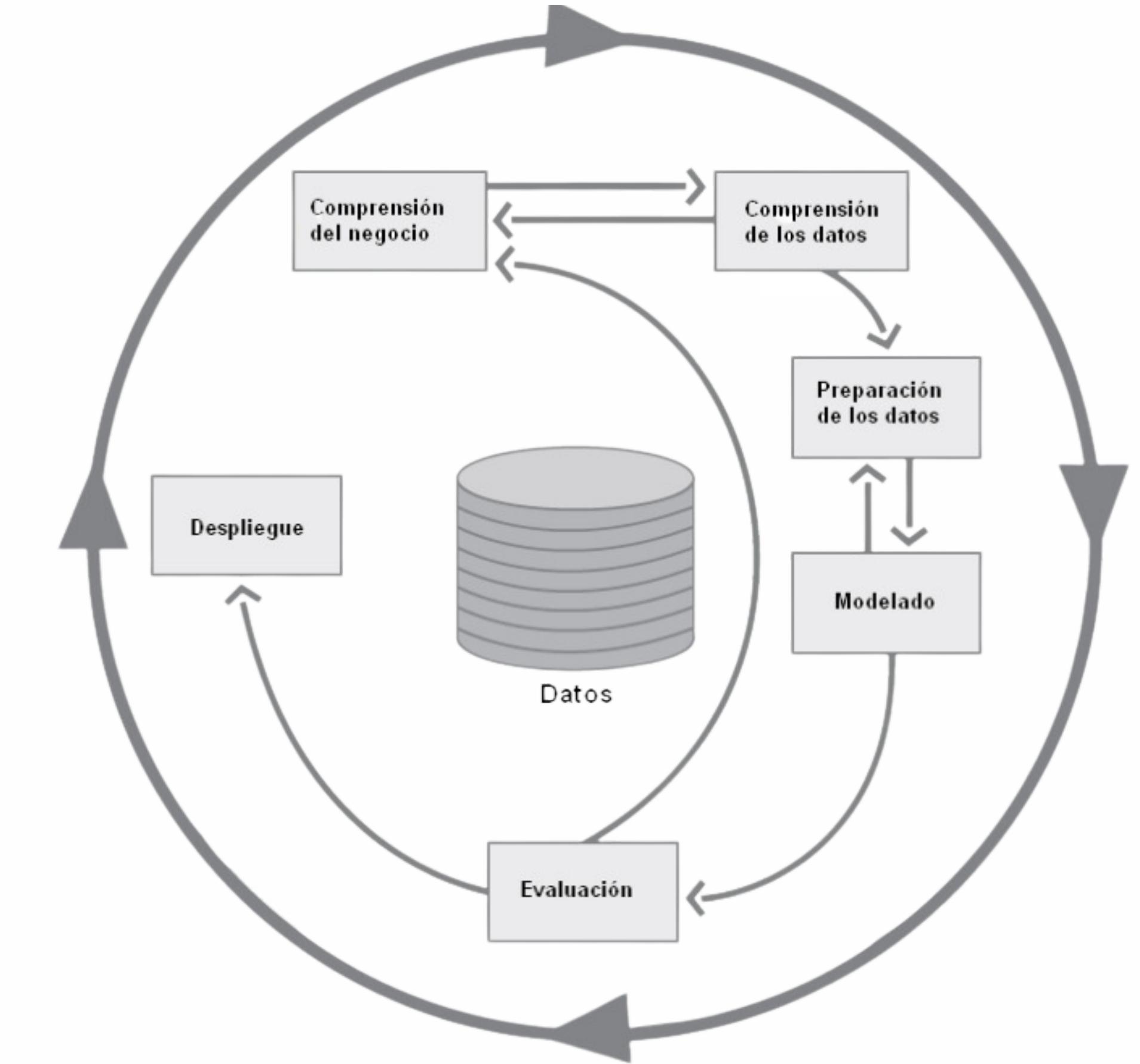


1. Objetivo principal

"Realizar un análisis de herramientas para optimizar el proceso de asignación de viajes de productos finales a diferentes transportistas proveedores de Ternium de acuerdo a las características de cada pedido."



Descripción metodología **CRISP-DM**





2. Fuentes de Datos

Descripción del conjunto de datos e integración

- 7 archivos en formato Excel
- 33 a 35 columnas c/u y entre 4695 a 9154 datos c/u
- Contienen varios valores nulos y algunos errores de ortografía.

Los archivos abarcan los meses de octubre, noviembre, diciembre, enero, febrero y marzo

Dimensión del dataset integrado y completo

- 33 columnas y contiene 33453 datos o filas

Dimensión del dataset integrado y limpio

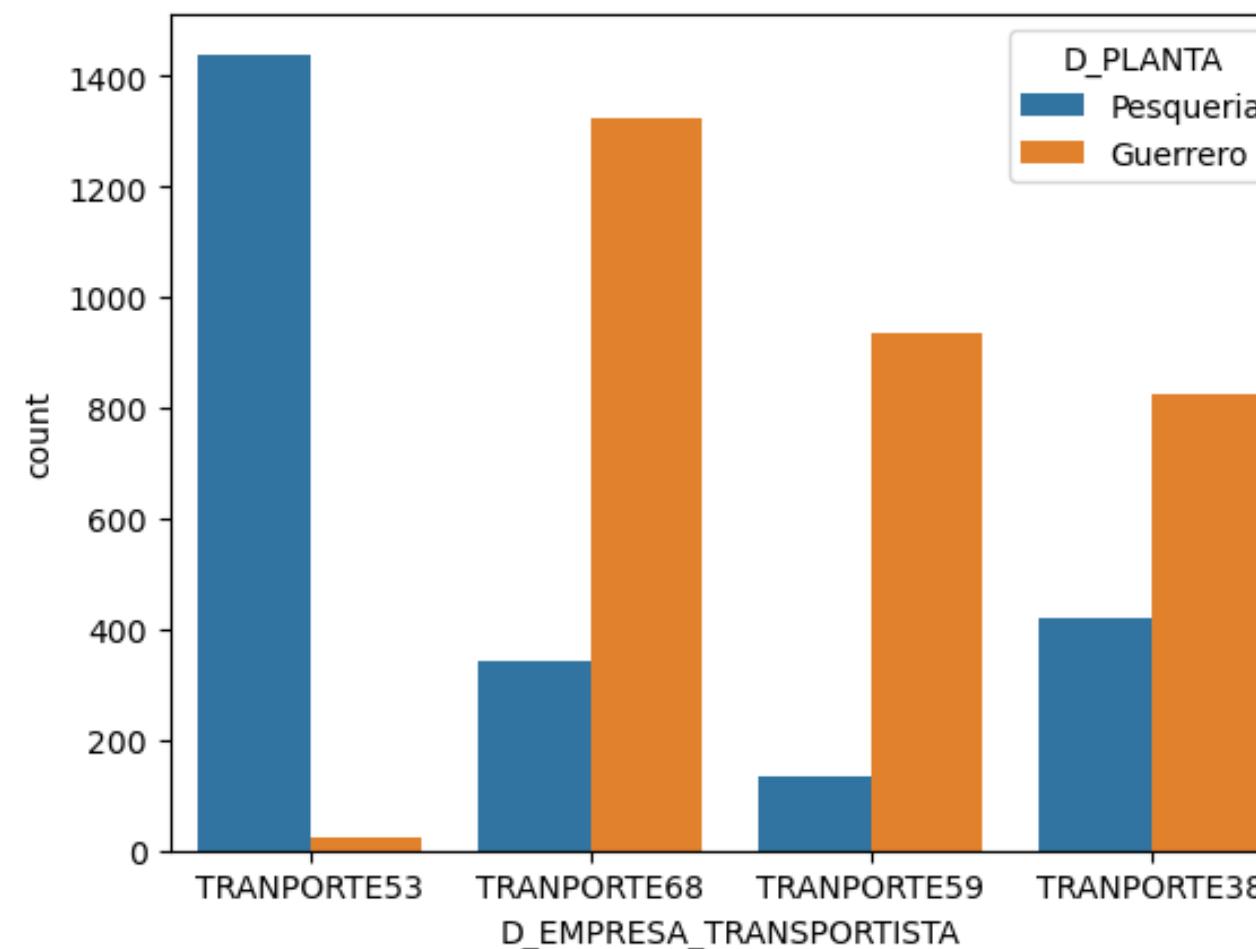
- 25 columnas y contiene 28689 datos o filas

Pandas

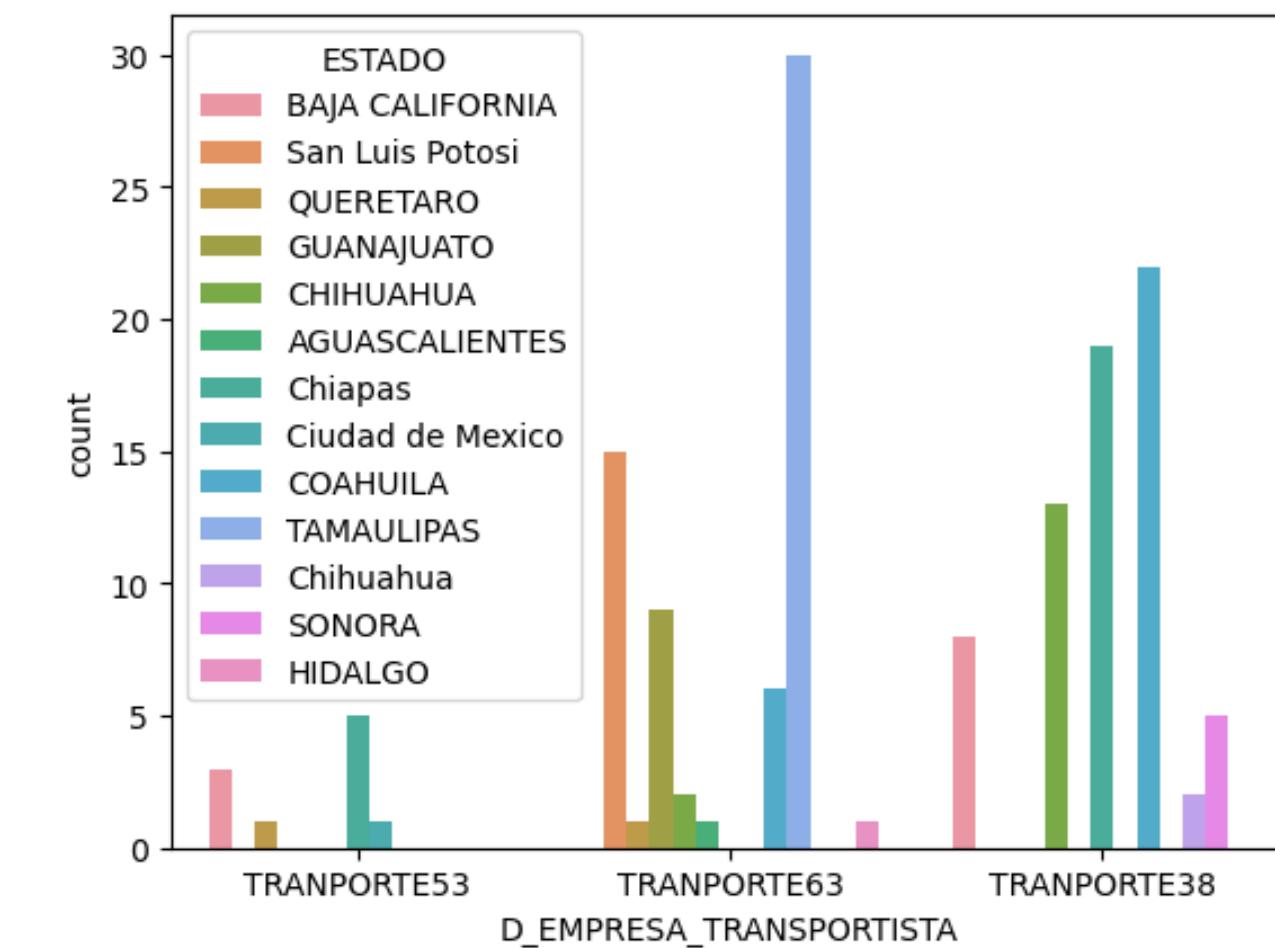


3 Exploración de Datos

Planta

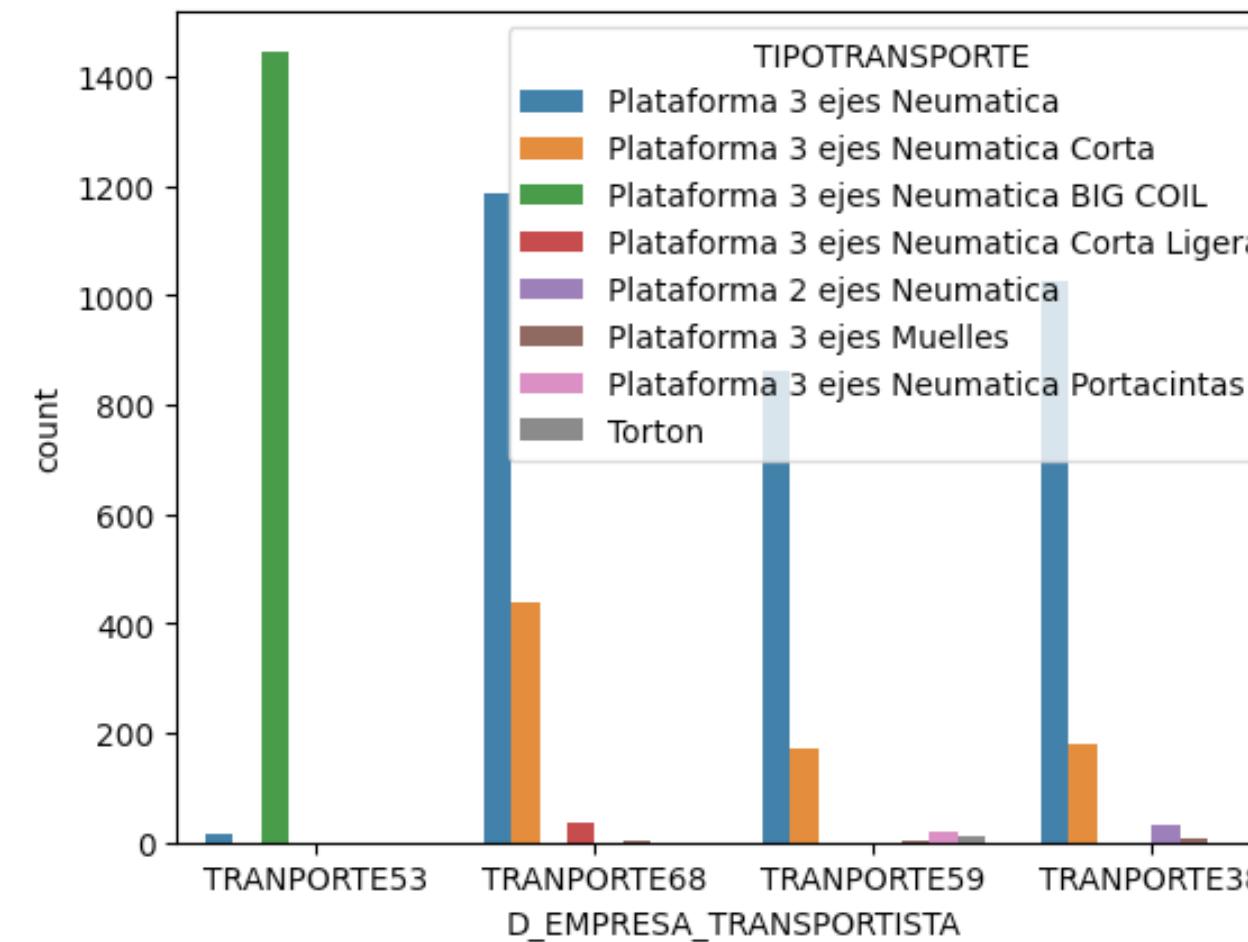


Estado

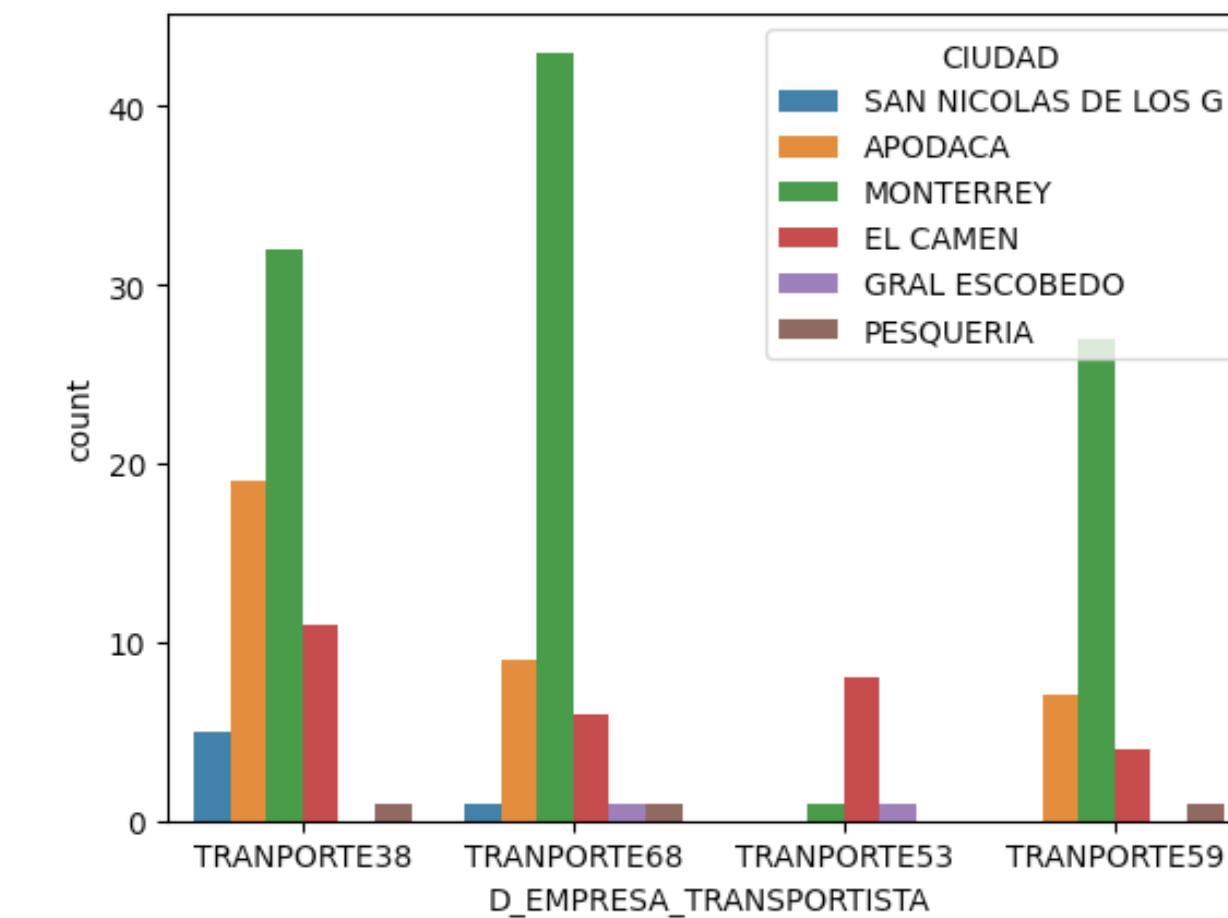


3 Exploración de Datos

Tipo Transporte

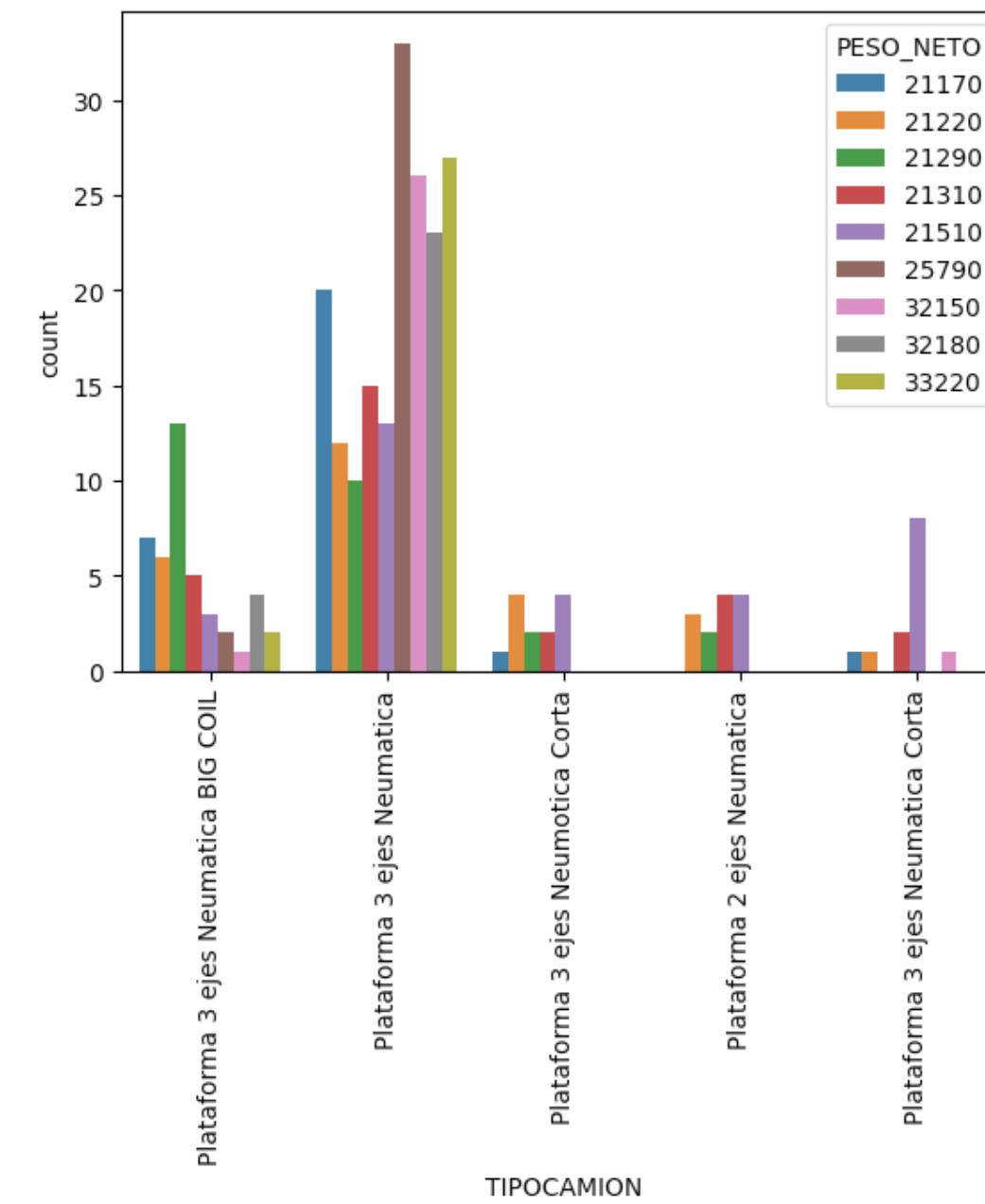


Ciudad

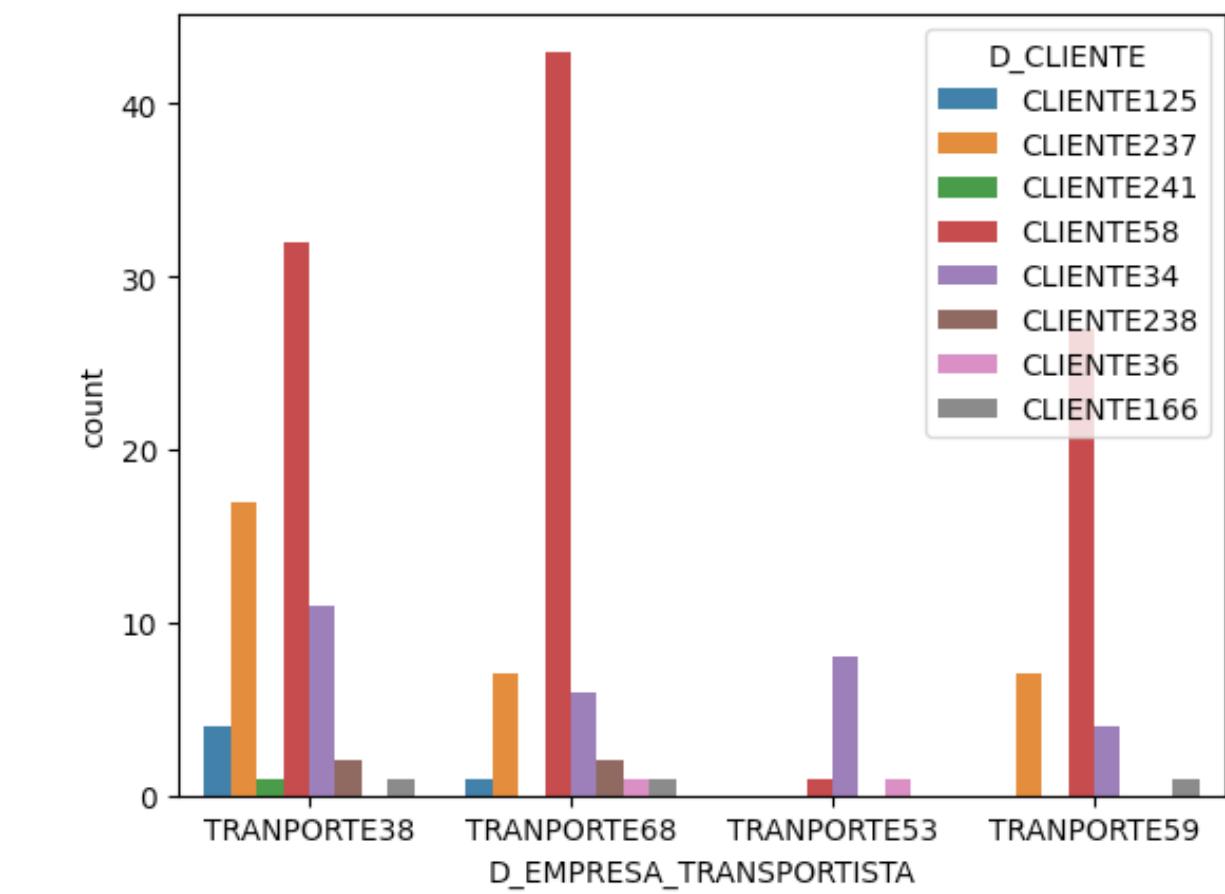


3 Exploración de Datos

Peso neto



Cliente



4 Generación y evaluación de modelos de aprendizaje

Selección variables predictoras, variable objetivo

```
[ ] # Separar las variables predictoras y la variable objetivo  
X = df_encoded[['D_CLIENTE', 'ESTADO','TIPOTRANSPORTE','PESO_NETO', 'CIUDAD', 'PLANTAORIGEN_Pesqueria']] # Variables predictoras numéricas  
y = df_encoded['D_EMPRESA_TRANSPORTISTA'] # Variable objetivo categórica
```

```
[ ] # Dividir los datos en conjuntos de entrenamiento y prueba  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

La división de entrenamiento y prueba en nuestros modelos iniciales sera de: 30/70

- Nuestras variables predictoras, seran:
- 'D_CLIENTE',
- 'ESTADO',
- 'TIPOTRANSPORTE',
- 'PESO_NETO',
- 'CIUDAD',
- 'PLANTAORIGEN_Pesqueria'

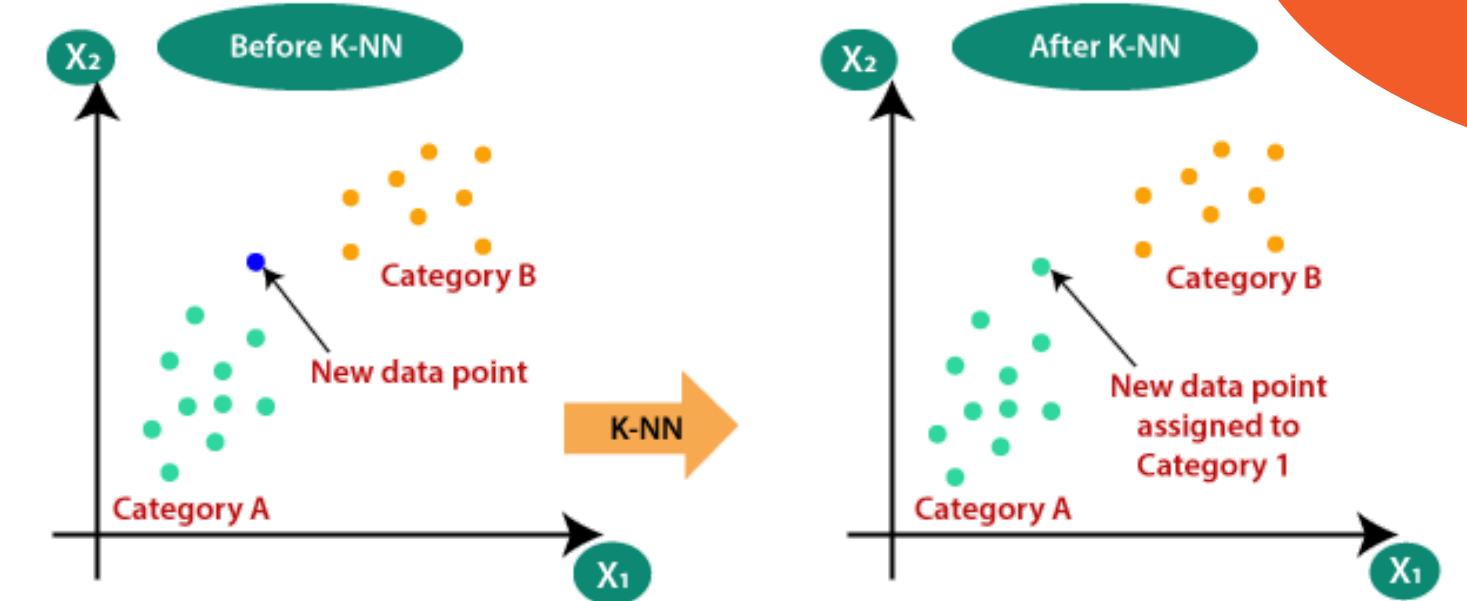
Nuestra variable objetivo sera obtener la empresa transportista;

'D_EMPRESA TRANPORTISTA'

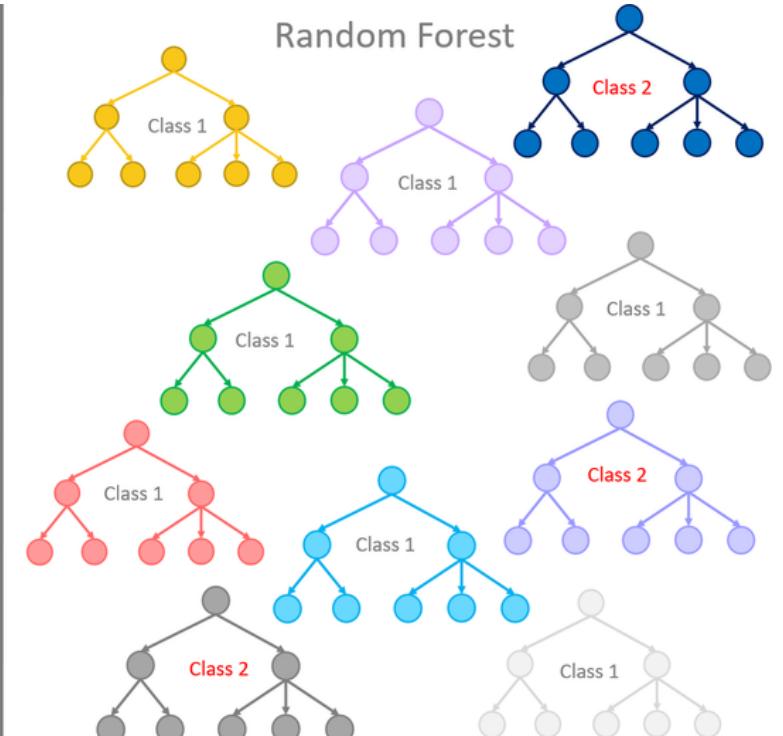
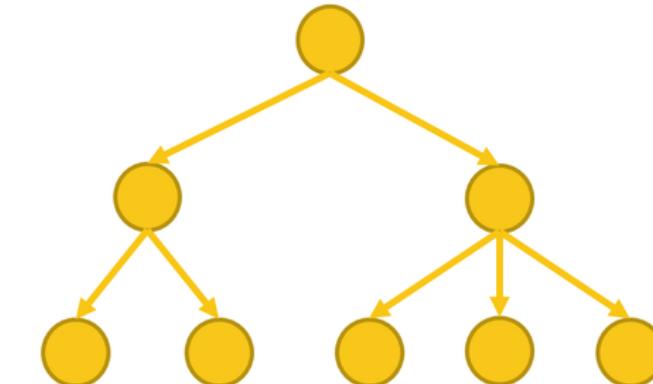
4 Generación y evaluación de modelos de aprendizaje

Modelos de clasificación

- **K-Nearest Neighbors (KNN)** calcula la distancia entre la instancia de prueba y sus k vecinos más cercanos en el espacio de características y asigna la etiqueta más común a la instancia de prueba.
- **Random Forest** utiliza un conjunto de árboles de decisión, donde cada árbol se entrena con una muestra aleatoria de datos y una selección aleatoria de características. Este modelo es conocido por su capacidad para manejar grandes conjuntos de datos y variables categóricas.
- **Decision Tree** se basa en una estructura de árbol para realizar predicciones. Cada nodo interno del árbol representa una característica, cada rama representa una posible salida. Este modelo es conocido por su capacidad para manejar datos faltantes y variables categóricas.



Single Decision Tree



4

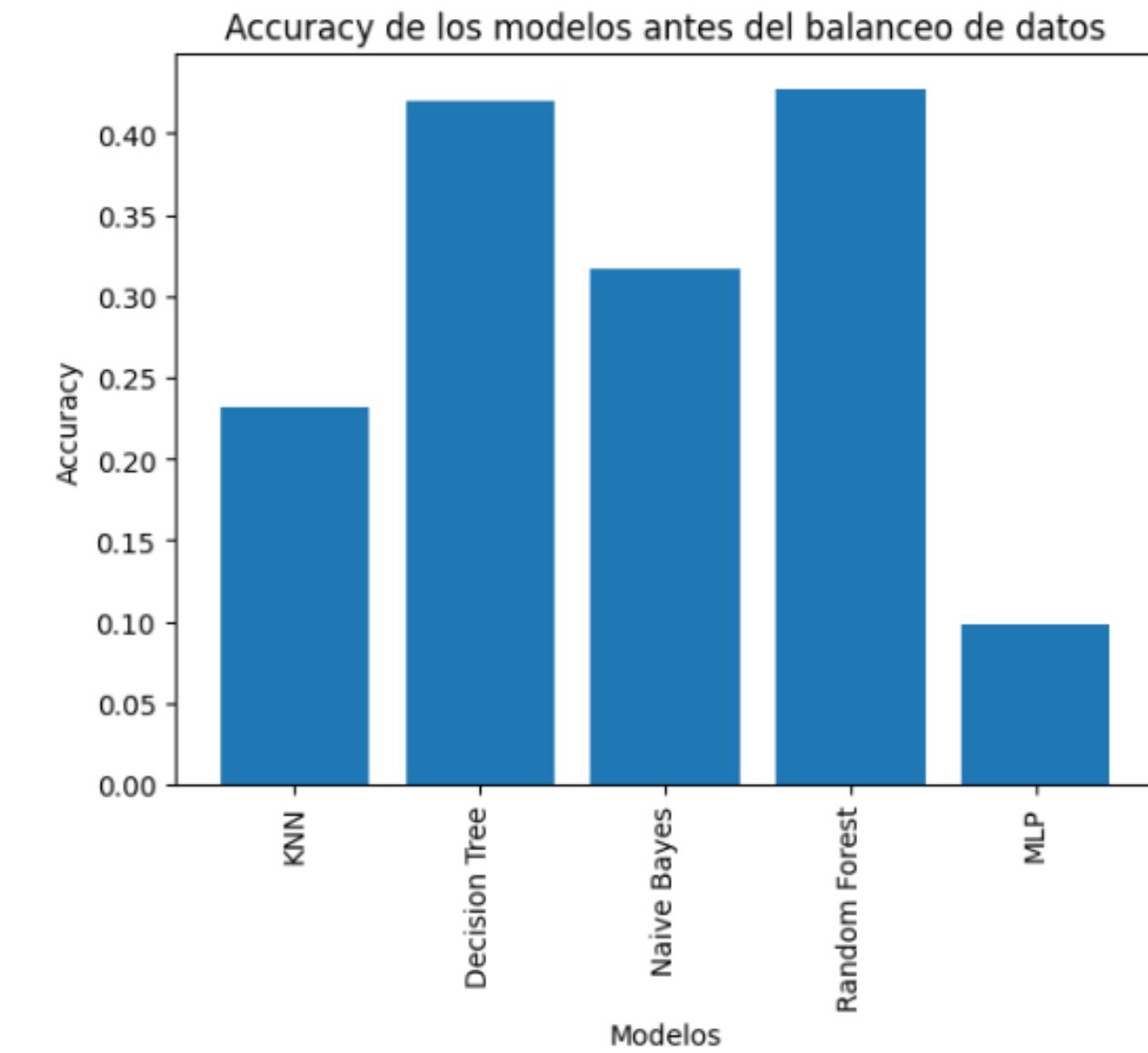
Generación y evaluación de modelos de aprendizaje

Problema Desbalance de los datos

```
df= pd.read_csv('https://raw.githubusercontent.com/kevingonzal/cienciaDatos/main/df_listo_modelo.csv')
df = df.loc[:, [ "PLANTAORIGEN_Pesqueria", "D_CLIENTE", "PESO_NETO","ESTADO","CIUDAD", "TIPOTRANSPORTE","D_EMPRESA_TRANSPORTISTA"]]
le = LabelEncoder()
df['D_EMPRESA_TRANSPORTISTA'] = le.fit_transform(df['D_EMPRESA_TRANSPORTISTA'])
df.rename(columns={"D_EMPRESA_TRANSPORTISTA": "target"}, inplace=True)
df.to_csv('df_modelos.csv', index=False)
```

	PLANTAORIGEN_Pesqueria	D_CLIENTE	PESO_NETO	ESTADO	CIUDAD	TIPOTRANSPORTE	target
0	1	187	30670	1	39	9	20
1	1	107	36890	8	35	5	50
2	1	145	23300	14	61	6	39
3	1	147	20480	22	1	5	15
4	1	99	36980	1	39	5	40
...
14403	0	229	34580	14	23	6	50
14404	0	131	19090	5	20	5	35
14405	0	131	19090	5	20	5	35
14406	0	131	19090	5	20	5	35
14407	0	205	22880	2	9	5	48

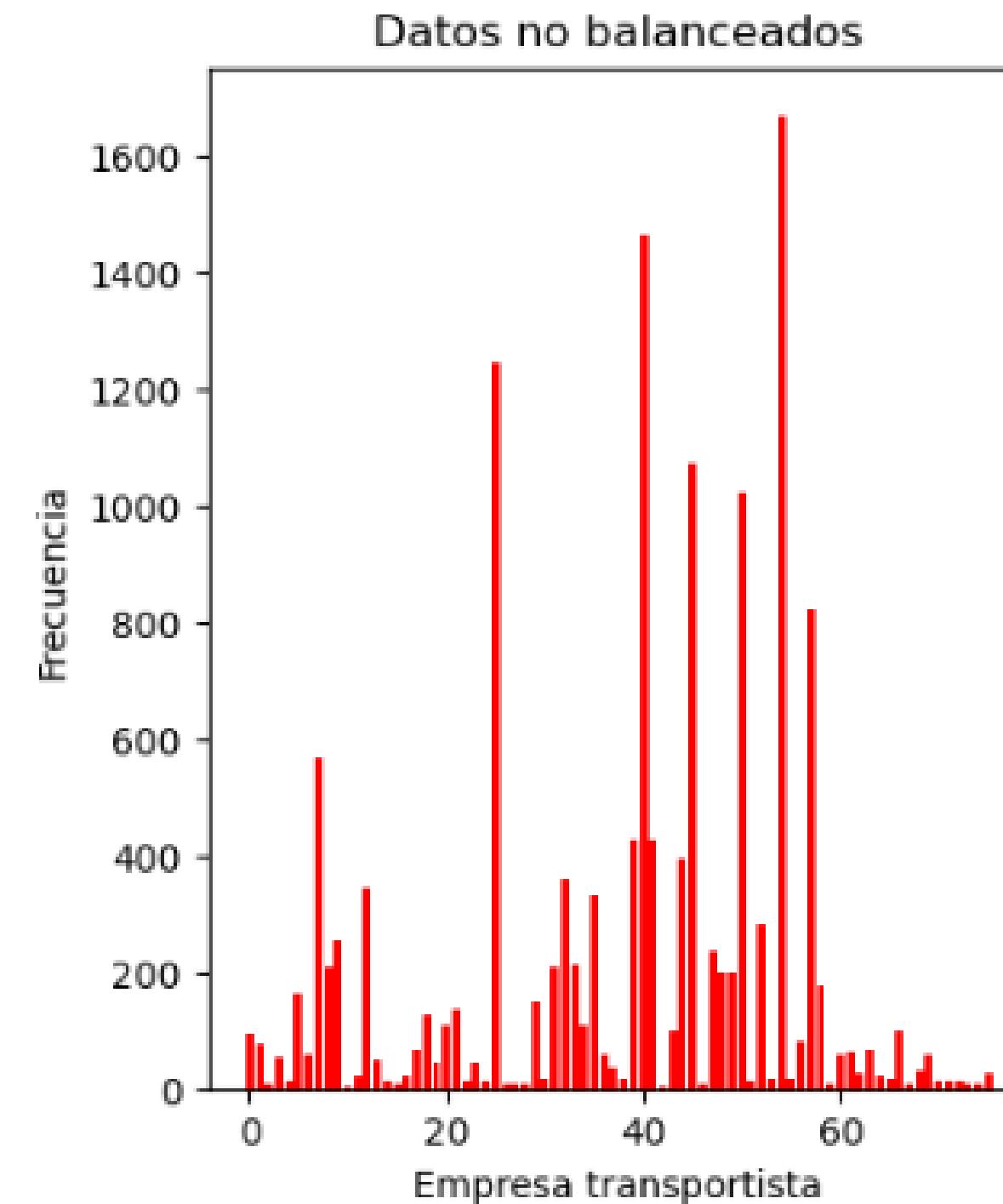
14408 rows × 7 columns



4

Generación y evaluación de modelos de aprendizaje

Desbalance de los datos



El desbalance de los datos es un problema común en los problemas de clasificación, que ocurre cuando la **cantidad de instancias** de cada clase en un problema multiclasé, **no es similar**.

Esto puede causar que los modelos de clasificación tengan una **precisión baja** en la **clase minoritaria**, ya que la mayoría de las veces están optimizados para maximizar la precisión global.

4 Generación y evaluación de modelos de aprendizaje

```
data = df.values
# split into input and output elements
x, y = data[:, :-1], data[:, -1]
# label encode the target variable
y = LabelEncoder().fit_transform(y)
# transform the dataset
oversample = SMOTE(k_neighbors=4)
x, y = oversample.fit_resample(x, y)
# summarize distribution
counter = Counter(y)
for k,v in counter.items():
    per = v / len(y) * 100
    print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
pyplot.bar(counter.keys(), counter.values())
pyplot.title('Datos balanceados')
pyplot.xlabel('Empresa transportista')
pyplot.ylabel('Frecuencia')
pyplot.show()
```

```
Class=50, n=1666 (1.316%)
Class=39, n=1666 (1.316%)
Class=15, n=1666 (1.316%)
Class=40, n=1666 (1.316%)
Class=41, n=1666 (1.316%)
Class=56, n=1666 (1.316%)
Class=54, n=1666 (1.316%)
Class=33, n=1666 (1.316%)
```

Método de sobremuestreo

- Este código realiza el balanceo de datos utilizando el método de sobremuestreo (**oversampling**) con la técnica SMOTE, la cual genera nuevos ejemplos sintéticos de la clase minoritaria.
- Primero, se divide el conjunto de datos en entradas (X) y la variable objetivo (y), y se aplica LabelEncoder para convertir las etiquetas de texto en valores numéricos.

4

Generación y evaluación de modelos de aprendizaje

Evaluación Modelos

```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear los modelos y entrenarlos con los datos de entrenamiento
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

nb = GaussianNB()
nb.fit(X_train, y_train)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)

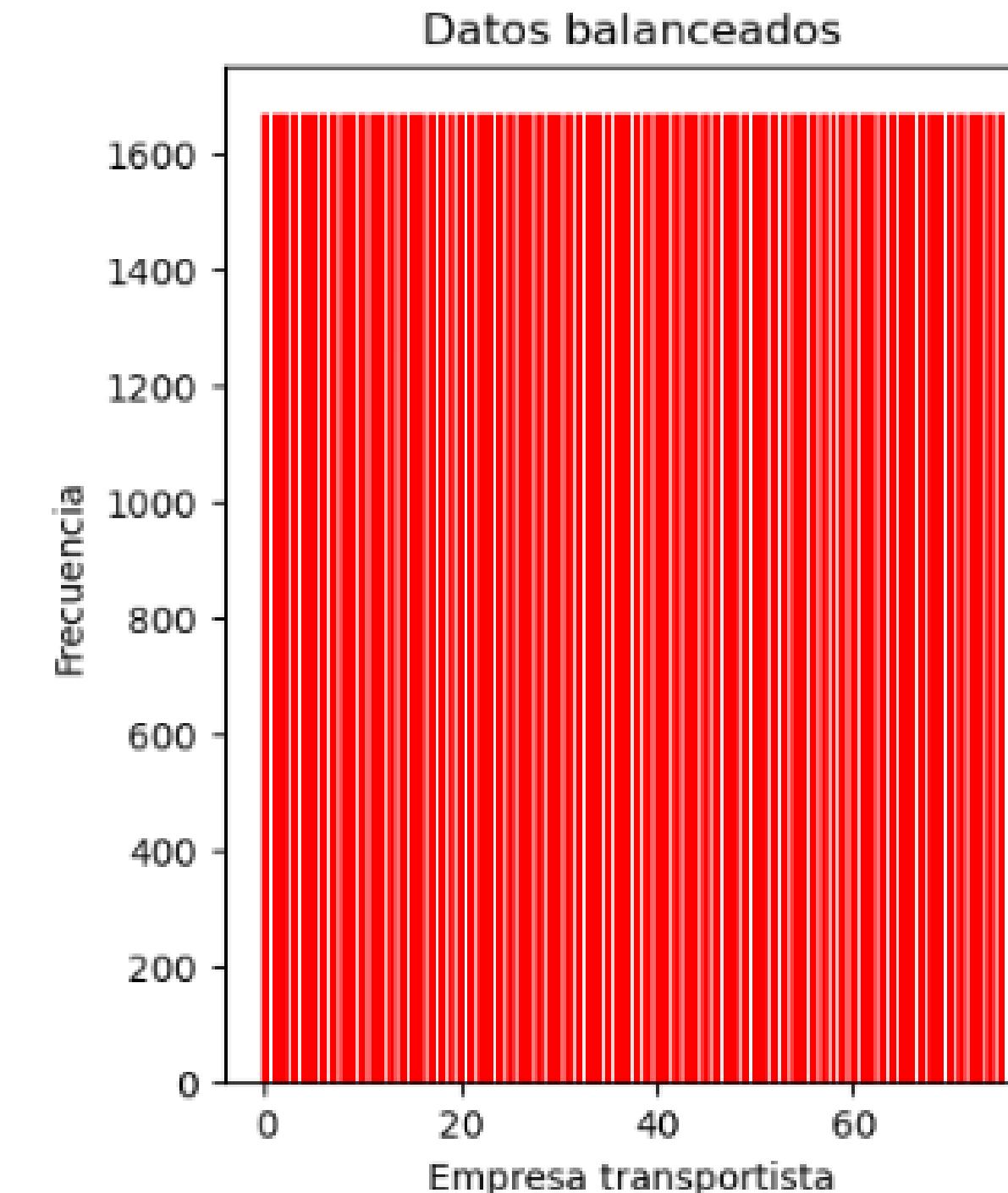
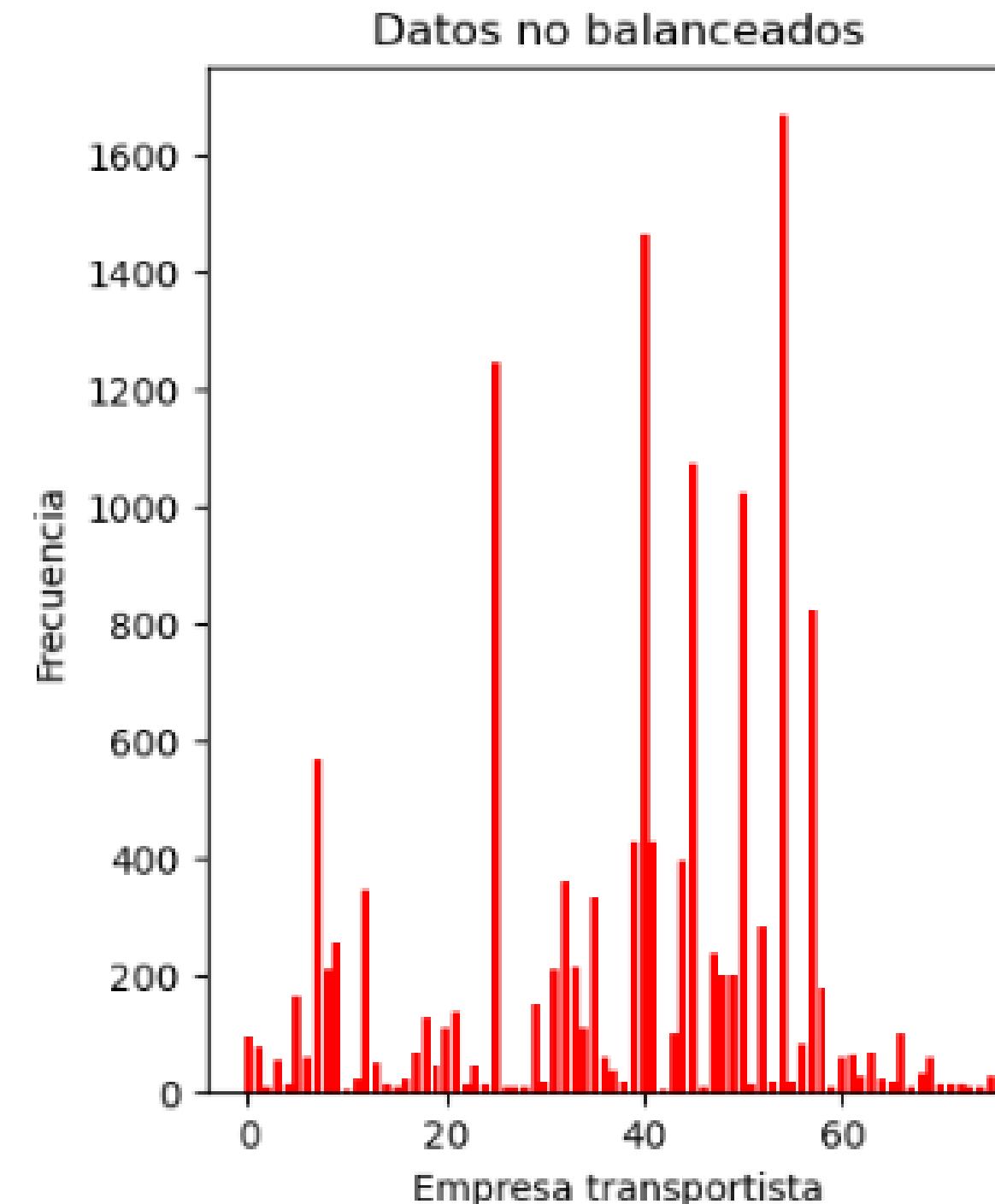
mlp = MLPClassifier()
mlp.fit(X_train, y_train)

# Evaluar el rendimiento de los modelos en los datos de prueba
acc_knn = knn.score(X_test, y_test)
acc_dt = dt.score(X_test, y_test)
acc_nb = nb.score(X_test, y_test)
acc_rf = rf.score(X_test, y_test)
acc_mlp = mlp.score(X_test, y_test)
```

En este modelo se realiza la evaluación de modelos, a través de volver a entrenar de nuevo nuestros modelos ya con los **datos balanceados** para poder comparar con respecto de los datos no balanceados, y ver como ha mejorado con respecto de una de sus métricas principales que es el **accuracy**.

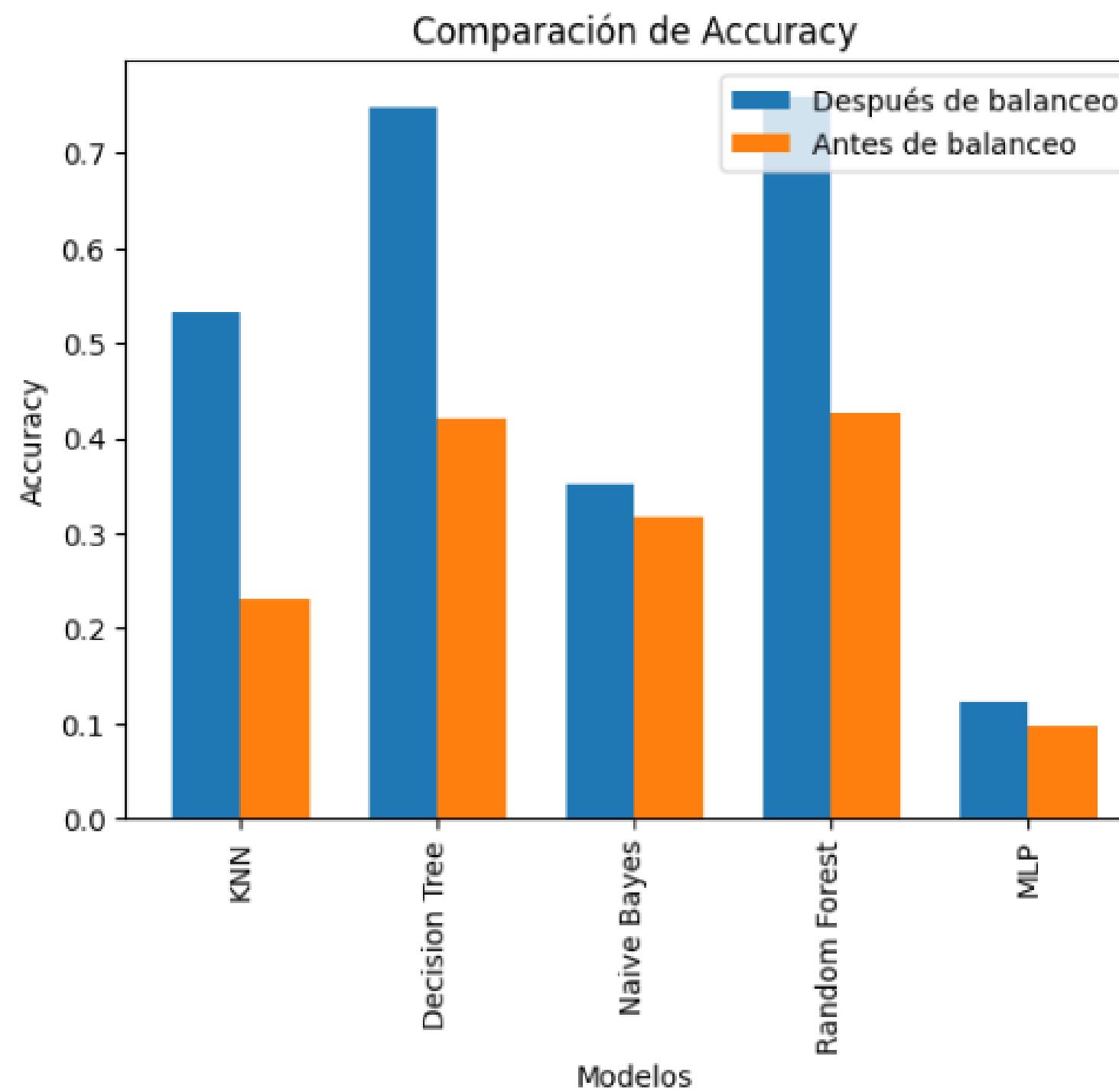
Generación y evaluación de modelos de aprendizaje

Comparativa de Datos Balanceados



Generación y evaluación de modelos de aprendizaje

Comparativa de Accuracy Datos Balanceados vs No Balanceados

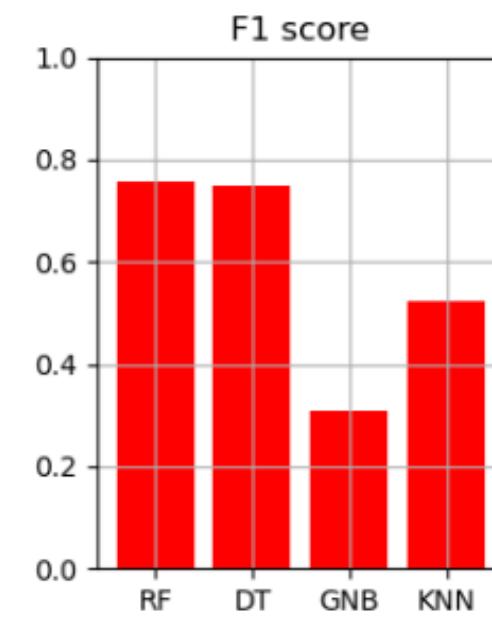
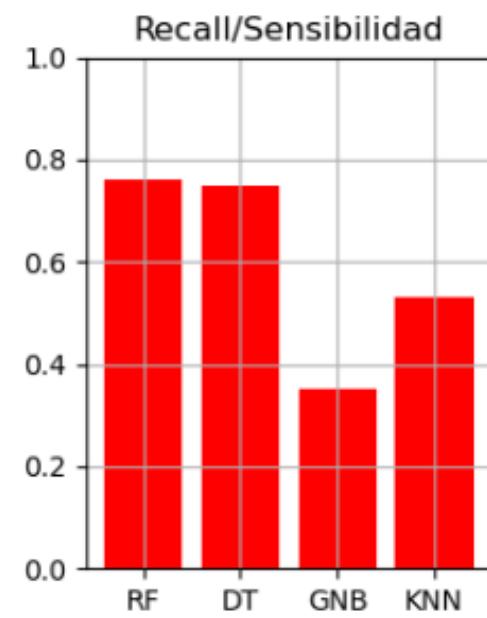
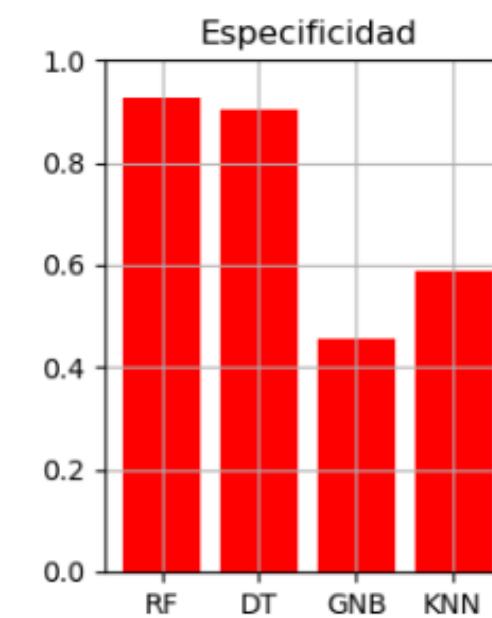
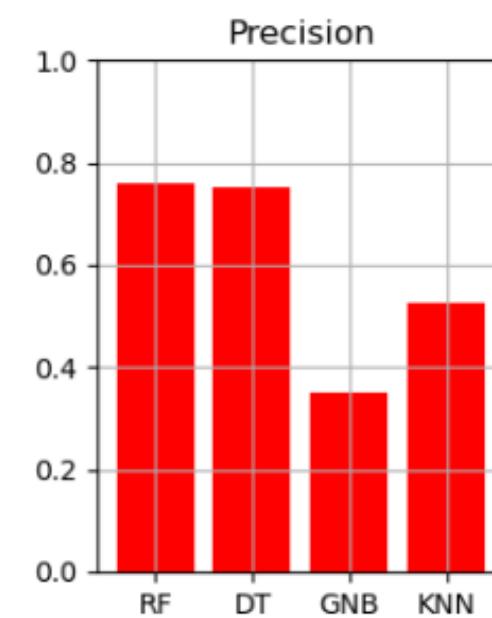
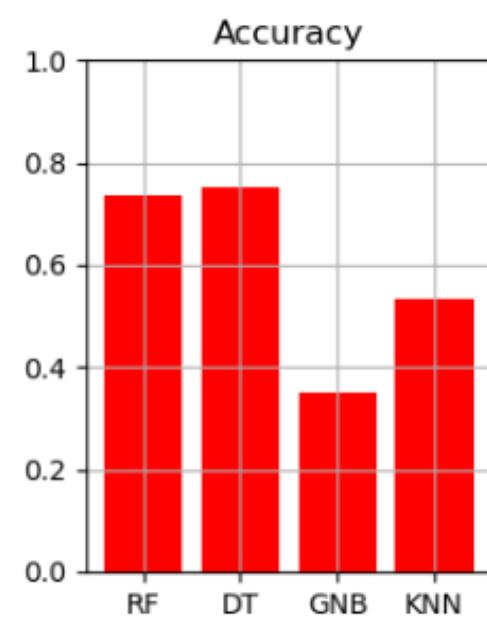


En este apartado se puede observar como ha mejorado el accuracy de todos los modelos en comparativa.

Sin embargo solo tomaremos de referencia los mejores modelos obtenidos hasta el momento, **Random Forest, Desicion Tree y KNN**.

Generación y evaluación de modelos de aprendizaje

Evaluación Métricas

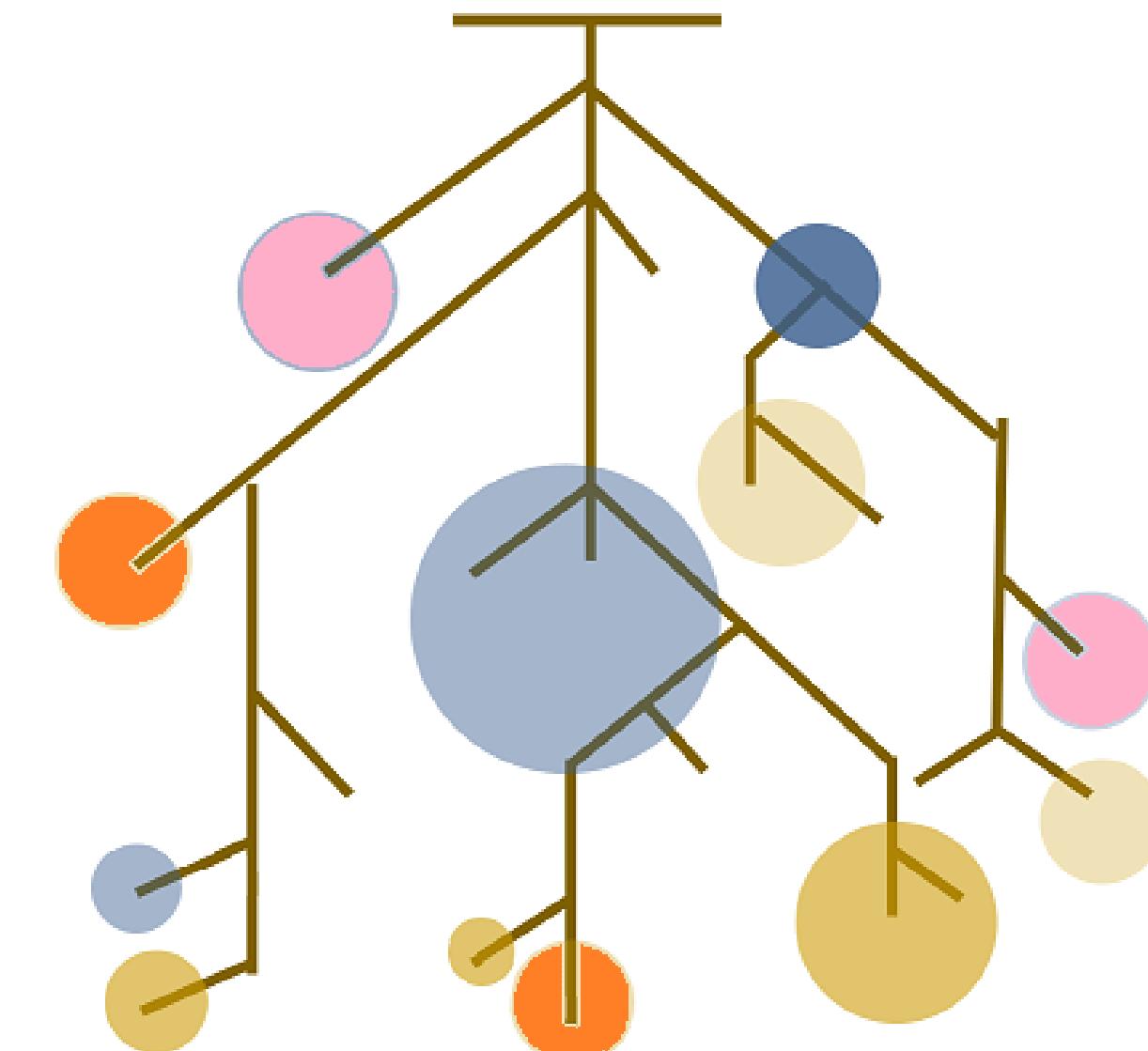


- **Exactitud (accuracy):** Es la proporción de predicciones correctas sobre el total de predicciones.
- **Precisión (precisión):** Es la proporción de verdaderos positivos (muestras correctamente clasificadas como positivas) sobre el total de muestras clasificadas como positivas.
- **Especificidad:** Es una métrica utilizada para evaluar el desempeño de un modelo de clasificación en términos de su capacidad para identificar correctamente las muestras negativas.
- **Sensibilidad (recall):** Es la proporción de verdaderos positivos sobre el total de muestras positivas en el conjunto de prueba.
- **F1-score:** Es una medida que combina la precisión y la sensibilidad del modelo.

4 Generación y evaluación de modelos de aprendizaje

Random Forest

El mejor modelo obtenido en comparativa con las métricas globales anteriores es '**Random Forest**', por lo que será el modelo de aprendizaje de clasificación que se utilizará como modelo de predicción con



5 Selección y despliegue del modelo

Ajuste hiperparámetros Bosques Aleatorios

```
: from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier()

# Definir los hiperparámetros que queremos ajustar
param_grid = {'n_estimators': [10, 50, 100, 150],
              'criterion': ['gini', 'entropy']}

# Crear el objeto GridSearchCV
grid_search = GridSearchCV(estimator = rfc, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2)

# Ajustar el objeto GridSearchCV a los datos
grid_search.fit(X, y)

# Imprimir los mejores parámetros encontrados
print(grid_search.best_params_)
```

Fitting 5 folds for each of 8 candidates, totalling 40 fits
{'criterion': 'gini', 'n_estimators': 150}

```
#Algoritmo Bosques Aleatorios
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
algoritmo = RandomForestClassifier(n_estimators = 100, criterion = 'entropy')
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
#Calculo la exactitud del modelo
accuracy = metrics.accuracy_score(y_test, y_pred)
accuracy
```

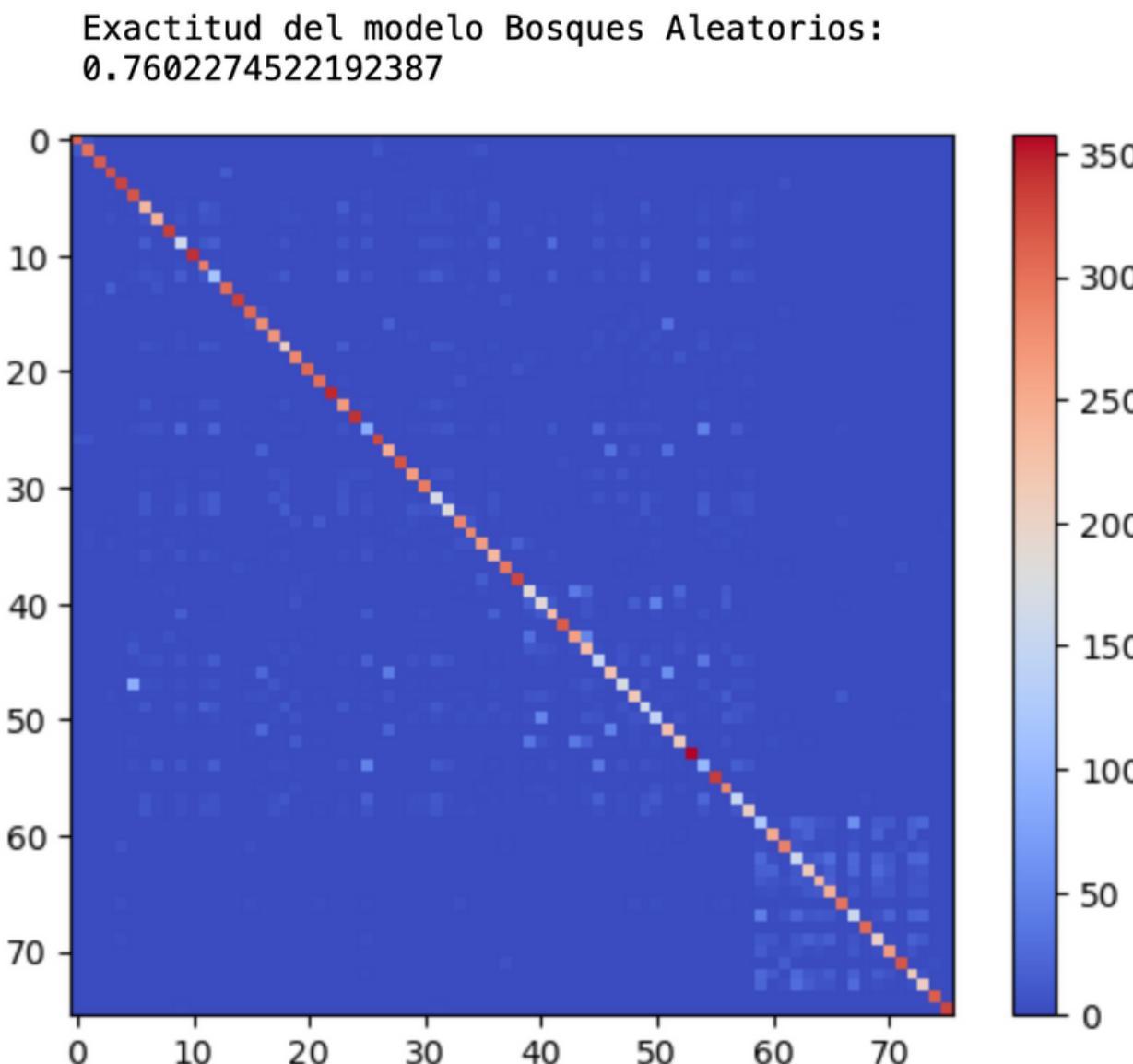
0.7617674932870004



```
#Algoritmo Bosques Aleatorios
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
algoritmo = RandomForestClassifier(n_estimators = 150, criterion = 'gini')
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
#Calculo la exactitud del modelo
accuracy = metrics.accuracy_score(y_test, y_pred)
accuracy
```

0.764452693097457

5 Selección y despliegue del modelo



```
: #Bosques Aleatorios
modelo = RandomForestClassifier(n_estimators = 150, criterion = 'gini')
modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test) # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]] #Obtener las 3 clases con mayores probabilidades

# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clase real
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print(f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')

for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1

#Evaluar el modelo
accuracy = aciertos / len(y_pred_classes)
m4=accuracy
print(f'Accuracy global: {accuracy:.2f}')

Instancia 25322: [49 36 6] - Resultado Real:36
Instancia 25323: [27 75 71] - Resultado Real:71
Instancia 25324: [58 29 50] - Resultado Real:50
Accuracy global: 0.90
```

5 Selección y despliegue del modelo

Ingreso de datos

```
leer_archivo = input("Por favor, escribe el nombre del archivo que quieras analizar: ")
archivo_predicciones=input("Por favor, escribe el nombre donde quieras guardar las predicciones: ")
```

Por favor, escribe el nombre del archivo que quieras analizar: prueba_1.csv
 Por favor, escribe el nombre donde quieras guardar las predicciones: predicciones_1.csv

Archivo que se desea predecir

```
prueba=pd.read_csv(leer_archivo)
x= prueba.shape[0]
prueba
```

	PLANTAORIGEN_Pesqueria	D_CLIENTE	PESO_NETO	ESTADO	CIUDAD	TIPOTRANSPORTE	target	EMPRESA_TRANSPORTISTA
0	0	227	38470	14	42		5	54
1	0	36	31530	14	13		5	54
2	0	227	39390	14	42		5	54
3	0	227	39850	14	42		5	54
4	0	78	31550	14	3		5	54

-
-  datos.csv
 -  prueba_1.csv
 -  Reto_final.ipynb

5 Selección y despliegue del modelo

```

predicciones = pd.DataFrame(), index=range(x), columns=['Prediccion1', 'Prediccion2', 'Prediccion3', 'Prediccion4',
for n in range(x):
    v1=prueba.iloc[n,0]
    v2=prueba.iloc[n,1]
    v3=prueba.iloc[n,2]
    v4=prueba.iloc[n,3]
    v5=prueba.iloc[n,4]
    v6=prueba.iloc[n,5]

    valores = [[v1,v2,v3,v4,v5,v6]]

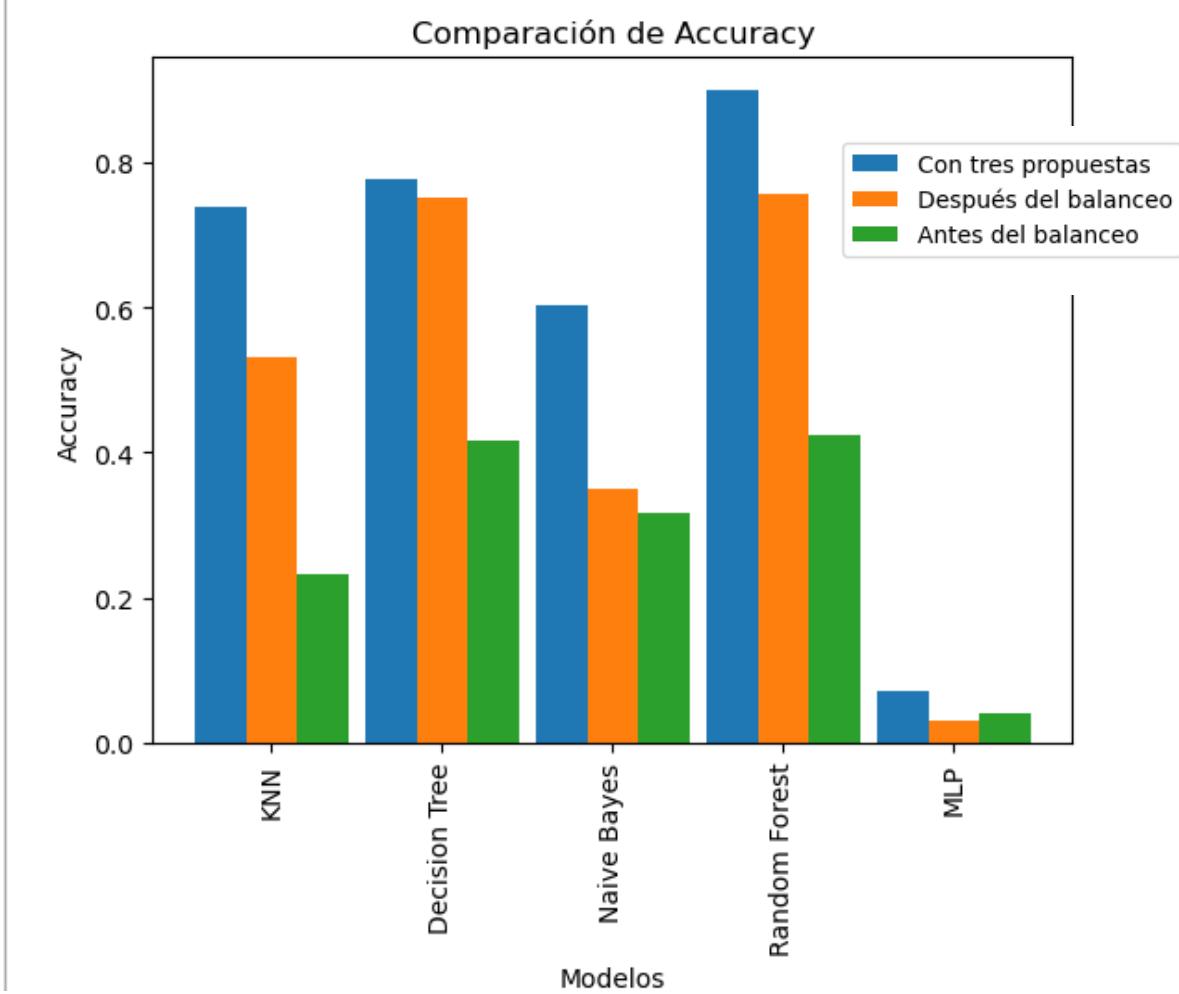
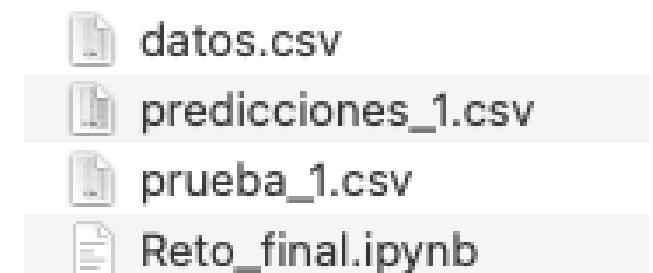
    probabilidades = algoritmo.predict_proba(valores)
    orden_probabilidades = np.argsort(probabilidades[0])[::-1]

    predicciones.iloc[n,0] =name[name["target"] == orden_probabilidades[0]].iloc[0,7]
    predicciones.iloc[n,1] =name[name["target"] == orden_probabilidades[1]].iloc[0,7]
    predicciones.iloc[n,2] =name[name["target"] == orden_probabilidades[2]].iloc[0,7]
    predicciones.iloc[n,3] =name[name["target"] == orden_probabilidades[3]].iloc[0,7]
    predicciones.iloc[n,4] =name[name["target"] == orden_probabilidades[4]].iloc[0,7]

prediccion= pd.concat([prueba, predicciones], axis=1)
prediccion.to_csv(archivo_predicciones, index=False)
prediccion

```

ESTADO	CIUDAD	TIPOTRANSPORTE	target	EMPRESA_TRANSPORTISTA	Prediccion1	Prediccion2	Prediccion3	Prediccion4	Prediccion5
14	42		5	54	TRANSPORTE68	TRANSPORTE68	TRANSPORTE59	TRANSPORTE49	TRANSPORTE4
14	13		5	54	TRANSPORTE68	TRANSPORTE68	TRANSPORTE59	TRANSPORTE45	TRANSPORTE47
14	42		5	54	TRANSPORTE68	TRANSPORTE68	TRANSPORTE3	TRANSPORTE4	TRANSPORTE31
14	42		5	54	TRANSPORTE68	TRANSPORTE68	TRANSPORTE9	TRANSPORTE4	TRANSPORTE31
14	3		5	54	TRANSPORTE68	TRANSPORTE68	TRANSPORTE20	TRANSPORTE16	TRANSPORTE19

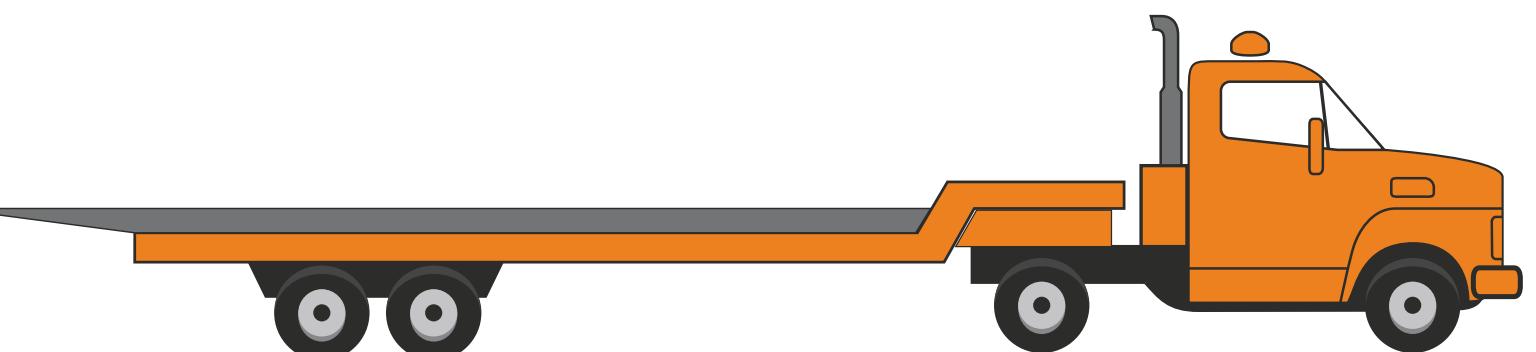


6 Conclusiones

Interpretación de los resultados obtenidos, impacto y utilidad

De acuerdo a los resultados, podemos aplicar el modelo de aprendizaje Random Forest para predecir los mejores transportistas a realizar el viaje.

Su impacto y utilidad será el eficientar la logística de las plantas en la hora de emparejar al transportista con el viaje asignado a realizar.



6 Resumen y recomendaciones

Resumen del proceso:

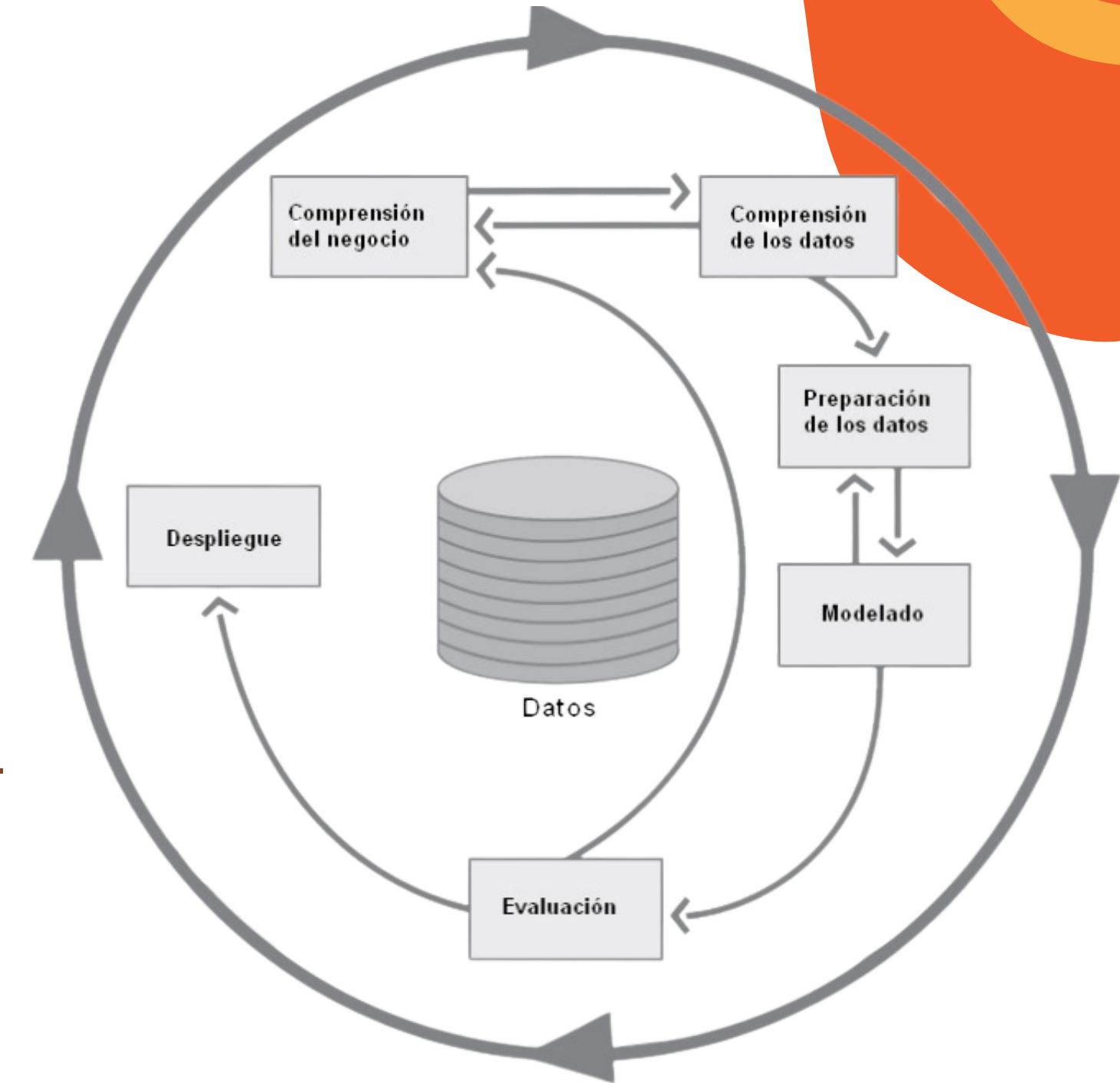
- Establecimiento del objetivo
- Integración de los datos
- Limpieza de datos
- Transformación de los datos
- Selección de variables más relevantes
- Aplicación de los modelos de aprendizaje supervisado
- Métricas de evaluación
- Despliegue de los resultados

Recomendaciones al socio formador:

- Un mayor énfasis al mantenimiento de las bases de datos.
- Eficientizar otros procesos de logística con modelos de aprendizaje supervisado.

Recomendaciones de pasos a seguir de este proyecto:

- Realizar una mejor interfaz gráfica que se adapte al empleado. Una app podría ser una excelente vía.



¡Gracias por su

atención!

Tiempo para preguntas