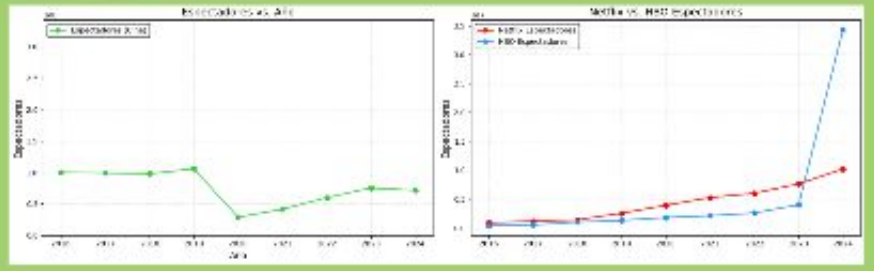
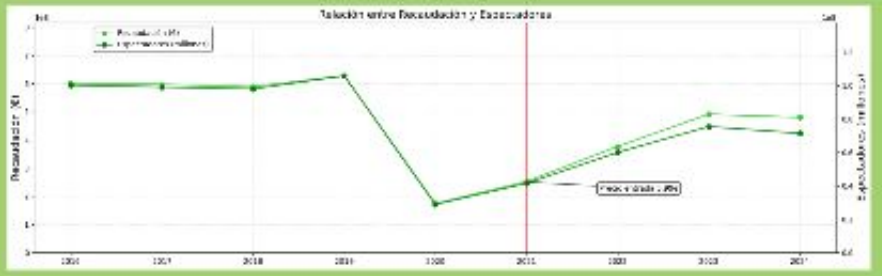
# Análisis del impacto de la pandemia y las plataformas streaming en el cine español

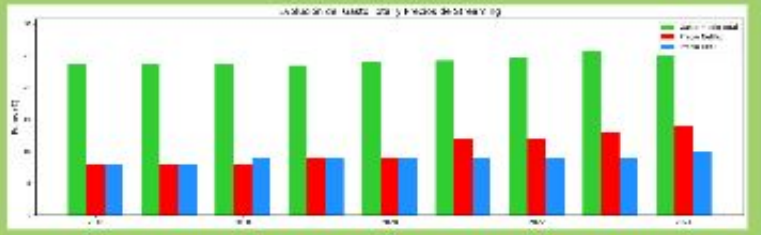
Datos extraídos de Kaggle, Ministerio de Cultura y análisis periodísticos (El País, Cope)

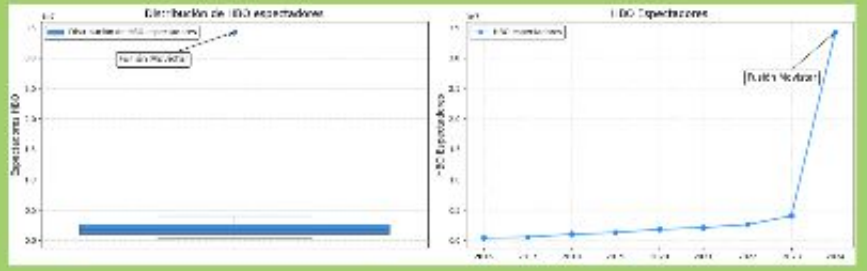
```
# ------------------------------------------------------------
# Datos obtenidos del Ministerio de Cultura y Arte, Kaggle
# y distintos análisis periodísticos (El País, Cope)
# ------------------------------------------------------------

# Crear el dataset de cine
cine_dataset = pd.DataFrame({
    "Año": [2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024],
    "Recaudación": [601375693, 597143256, 588675481, 628076776, 174153477, 252069251, 376814060, 491555176, 479518962],
    "Espectadores": [100270569, 99132198, 97993827, 105938534, 28876970, 41499889, 59995576, 75472359, 71437087],
    "Fiesta del cine": [2600000, 2300000, 1800000, 2350000, None, 1200000, 704977, 1198062, 1394058],
    "Pantallas": [3545, 3585, 3578, 3674, 3585, 3633, 3650, 3609, 3464],
    "Gasto medio/espectador": [5.93, 5.93, 5.93, 5.86, 5.99, 6.04, 6.19, 6.42, 6.24],
    "Frecuencia": [2.2, 2.2, 2.09, 2.23, 0.56, 0.86, 1.24, 1.6, 1.52],
    "Nº Películas": [1797, 1869, 1951, 1862, 1586, 2105, 2543, 2450, 2357],
    "Netflix espectadores": [900000, 1163000, 1400000, 2550000, 3900000, 5256064, 6000000, 7620000, 10200000],
    "HBO espectadores": [414000, 538000, 1000000, 1290000, 1800000, 2100000, 2600000, 4000000, 34260000],
    "Netflix precio": [7.99, 7.99, 7.99, 8.99, 8.99, 11.99, 11.99, 12.99, 13.99],
    "HBO precio": [7.99, 7.99, 8.99, 8.99, 8.99, 8.99, 8.99, 8.99, 9.99]
})


# Crear el dataset de ventas de tickets
cinema_hall_ticket_sales = pd.DataFrame({
    "Edad": [25, 30, 22, 34, 40, 45, 28, 32, 41, 37] * 300,  # Más de 2000 filas
    "Género película": ["Acción", "Drama", "Comedia", "Acción", "Terror", "Acción", "Drama", "Comedia", "Terror", "Acción"] * 300,
    "Asiento": ["VIP", "Normal", "VIP", "Normal", "Normal", "Normal", "VIP", "Normal", "Normal", "VIP"] * 300,
    "Nº de personas": [1, 2, 1, 3, 4, 2, 1, 3, 2, 4] * 300
})

# Manejar rutas del sistema
data_dir = Path("data")
data_dir.mkdir(parents=True, exist_ok=True)

repeated_cine_dataset = pd.DataFrame(
    np.tile(cine_dataset.values, (len(cinema_hall_ticket_sales) // len(cine_dataset) + 1, 1)),
    columns=cine_dataset.columns
)

# Ajustar el tamaño
repeated_cine_dataset = repeated_cine_dataset.iloc[:len(cinema_hall_ticket_sales)].reset_index(drop=True)

# Combinar ambos datasets
df = pd.concat([cinema_hall_ticket_sales.reset_index(drop=True), repeated_cine_dataset], axis=1)

# Verificar tamaño del dataset combinado
print(f"El dataset combinado tiene {df.shape[0]} filas y {df.shape[1]} columnas.")

# Guardar el dataset combinado
combined_dataset_path = data_dir / "combined_dataset.csv"
df.to_csv(combined_dataset_path, index=False)
print(f"El dataset combinado ha sido guardado como '{combined_dataset_path}'.")
```

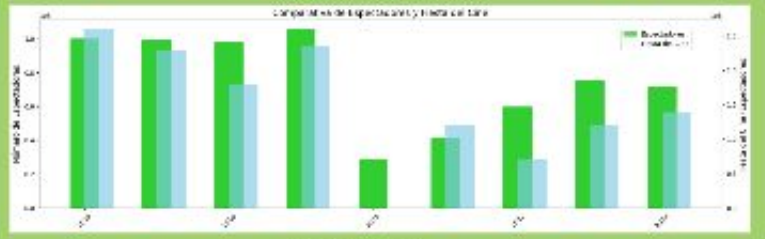
```
# Renombrar columnas (a español) de cinema_hall_ticket_sales
cinema_hall_ticket_sales.rename(columns={
    'Age': 'Edad',
    'Movie_Genre': 'Género película',
    'Seat_Type': 'Asiento',
    'Number_of_Person': 'Nº de personas',
    'Purchase_Again': 'Compró de nuevo'
}, inplace=True)

# Limpiar valores NaN (2020) y ajustar tipos
df['Fiesta del cine'] = df['Fiesta del cine'].fillna(0).astype(int)

# Ajustar valores (cambiar Alone a 1)
df['Nº de personas'] = df['Nº de personas'].replace('Alone', 1)

# Crear variables
df['Pandemic_Impact'] = df['Año'].apply(lambda x: 1 if x == 2020 else 0)
df['Streaming_Impact'] = df['Año'].apply(lambda x: 1 if x >= 2016 else 0)
df['Cost_Per_Spectator'] = df['Recaudación'] / df['Espectadores']

# Crear columnas binarias específicas basadas en la columna 'Género película'
df['Género Terror'] = (df['Género película'] == 'Terror').astype(int)
df['Género Comedia'] = (df['Género película'] == 'Comedia').astype(int)
df['Género Acción'] = (df['Género película'] == 'Acción').astype(int)
df['Género Drama'] = (df['Género película'] == 'Drama').astype(int)

# Borrar duplicados
df = df.drop_duplicates()

# Dataset listo para análisis
print("Dataset enriquecido y preparado para análisis.")
```

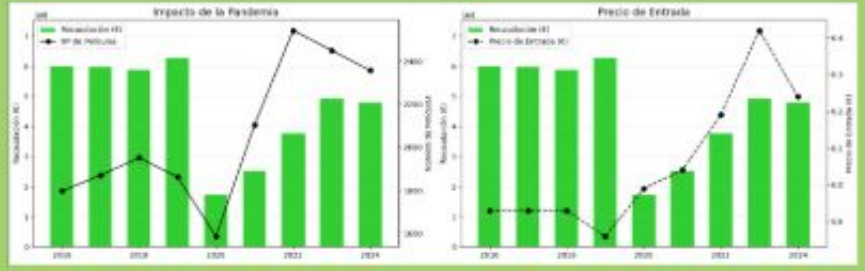



Relación entre Recaudación y Espectadores
Precio de Entrada ($5)



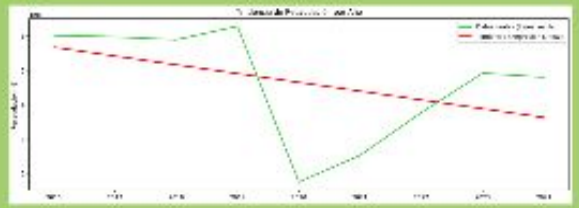

Distribución de HBO espectadores
Punto de Inflexión
HBO Espectadores
Punto Neutral




Impacto de la Pandemia
Precio de Entrada


Proyección Futura del Cine
Proyección Futura

```
# ---------------------------
# Regresión Lineal Simple
# ---------------------------
X_simple = df[['Año']]
y = df['Recaudación']

# Dividir en conjuntos de entrenamiento y prueba (80%/20%)
X_train, X_test, y_train, y_test = train_test_split(X_simple, y, test_size=0.2, random_state=42)

# Crear y entrenar el modelo de regresión lineal
reg_model = LinearRegression()
reg_model.fit(X_train, y_train)

# Predecir en el conjunto de prueba
y_pred = reg_model.predict(X_test)

plt.figure(figsize=(16, 5))
plt.plot(X_simple, y, color='#32CD32', linewidth=2, label='Datos reales (Línea verde)')
plt.plot(X_simple, reg_model.predict(X_simple), color='red', linestyle='-', linewidth=2,
    label='Tendencia (Regresión Lineal)')
plt.ylabel("Recaudación (€)")
plt.title("Tendencia de Recaudación por Año")
plt.legend()
plt.show()
```



```
# ---------------------------
# Regresión Lineal Multivariada
# ---------------------------
lr_model = LinearRegression()
lr_model.fit(X_train_reg, y_train_reg)
y_pred_lr = lr_model.predict(X_test_reg)

mae_lr = mean_absolute_error(y_test_reg, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test_reg, y_pred_lr))
r2_lr = r2_score(y_test_reg, y_pred_lr)

print("Regresión Lineal:")
print(f"MAE: {mae_lr:.2f}")
print(f"RMSE: {rmse_lr:.2f}")
print(f"R²: {r2_lr:.2f}")
```

Regresión Lineal:
MAE: 719691.03
RMSE: 867428.75
R²: 1.00

```
# ---------------------------
# Random Forest Regression
# ---------------------------
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train_reg, y_train_reg)
y_pred_rf = rf_model.predict(X_test_reg)

mae_rf = mean_absolute_error(y_test_reg, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test_reg, y_pred_rf))
r2_rf = r2_score(y_test_reg, y_pred_rf)

print("Random Forest Regression:")
print(f"MAE: {mae_rf:.2f}")
print(f"RMSE: {rmse_rf:.2f}")
print(f"R²: {r2_rf:.2f}")

print("\nImportancia de variables en Random Forest:")
for feature, importance in zip(features_reg, rf_model.feature_importances_):
    print(f"{feature}: {importance:.4f}")

# Graficar Predicciones vs Valores Reales
plt.figure(figsize=(16, 5))
plt.scatter(y_test_reg, y_pred_rf, color='#32CD32', label='Predicciones vs Valores Reales')
plt.plot([y_test_reg.min(), y_test_reg.max()], [y_test_reg.min(), y_test_reg.max()], 'k--', linewidth=2, label='Línea Ideal')
plt.xlabel('Valores Reales', fontsize=12)
plt.ylabel('Predicciones', fontsize=12)
plt.title('Predicciones vs Valores Reales - Random Forest Regression')
plt.legend()
plt.tight_layout()
plt.show()
```

```
# ---------------------------------------
# Validación cruzada para Random Forest
# ---------------------------------------
cv_scores_rf = cross_val_score(best_rf, X_reg, y_reg, cv=10, scoring="neg_mean_absolute_error")
mean_mae_rf = np.mean(np.abs(cv_scores_rf))

# Métricas del modelo
mae_best_rf = mean_absolute_error(y_test_reg, y_pred_best_rf)
rmse_best_rf = np.sqrt(mean_squared_error(y_test_reg, y_pred_best_rf))
r2_best_rf = r2_score(y_test_reg, y_pred_best_rf)

print(f"MAE promedio (Validación cruzada): {mean_mae_rf:.2f}")
print("Mejores Hiperparámetros:")
print(grid_search.best_params_)
```

```
# ---------------------------------
# GridSearchCV para Random Forest
# ---------------------------------
param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5]
}
# Configurar GridSearchCV con validación estratificada
grid_search = GridSearchCV(
    RandomForestRegressor(random_state=42),
    param_grid,
    cv=3,
    scoring="r2",
    n_jobs=-1,
    error_score="raise"
)
# Ajustar el modelo
grid_search.fit(X_train_reg, y_train_reg)

# Resultados del mejor modelo
best_rf = grid_search.best_estimator_
y_pred_best_rf = best_rf.predict(X_test_reg)

# Métricas del modelo optimizado
print("Mejor Random Forest (por GridSearchCV):")
print(f"MAE: {mean_absolute_error(y_test_reg, y_pred_best_rf):.2f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test_reg, y_pred_best_rf)):.2f}")
print(f"R²: {r2_score(y_test_reg, y_pred_best_rf):.4f}")

print("Mejores Hiperparámetros:")
print(grid_search.best_params_)
```

```
# ----------------------
# XGBoost Regression
# ----------------------
# Definir el modelo XGBoost
xgb_model = XGBRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)

# Entrenar el modelo y predecir
xgb_model.fit(X_train_reg, y_train_reg)
y_pred_xgb = xgb_model.predict(X_test_reg)

mae_xgb = mean_absolute_error(y_test_reg, y_pred_xgb)
rmse_xgb = np.sqrt(mean_squared_error(y_test_reg, y_pred_xgb))
r2_xgb = r2_score(y_test_reg, y_pred_xgb)

print("XGBoost Regression:")
print(f"MAE: {mae_xgb:.2f}")
print(f"RMSE: {rmse_xgb:.2f}")
print(f"R²: {r2_xgb:.4f}")

# Validación cruzada para XGBoost Regression
cv_scores_xgb = cross_val_score(
    xgb_model,
    X_reg,
    y_reg,
    cv=10,
    scoring='neg_mean_absolute_error'
)
# Calcular el MAE promedio
mean_mae_xgb = np.mean(np.abs(cv_scores_xgb))
print("XGBoost Regression - MAE promedio (10 folds):", mean_mae_xgb)
```

```
XGBoost Regression:
MAE: 192.72
RMSE: 218.71
R²: 1.0000
XGBoost Regression - MAE promedio (10 folds): 196.17777777777778
```

```
# ----------------------
# Ridge Regression (L2)
# ----------------------
# Escalar los Datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_reg)
X_test_scaled = scaler.transform(X_test_reg)

# Ridge Regression con Alpha = 0.01
ridge_best = Ridge(alpha=0.01)
ridge_best.fit(X_train_scaled, y_train_reg)
y_pred_ridge_best = ridge_best.predict(X_test_scaled)

mae_ridge = mean_absolute_error(y_test_reg, y_pred_ridge_best)
rmse_ridge = np.sqrt(mean_squared_error(y_test_reg, y_pred_ridge_best))
r2_ridge = r2_score(y_test_reg, y_pred_ridge_best)

print("Ridge Regression (Alpha = 0.01):")
print(f"MAE: {mae_ridge:.2f}")
print(f"RMSE: {rmse_ridge:.2f}")
print(f"R²: {r2_ridge:.4f}")

# Validación Cruzada para Alpha = 0.01
cv_scores_ridge = cross_val_score(ridge_best, scaler.transform(X_reg), y_reg, cv=10, scoring='neg_mean_absolute_error')
mean_mae_cv = np.mean(np.abs(cv_scores_ridge))
print("Ridge Regression - MAE promedio (Validación cruzada con Alpha = 0.01):", mean_mae_cv)
```

```
Ridge Regression (Alpha = 0.01):
MAE: 571611.47
RMSE: 849991.64
R²: 1.0000
Ridge Regression - MAE promedio (Validación cruzada con Alpha = 0.01): 1171974.2782223548
```

```
# ----------------------------------
# Reducción de Dimensionalidad: PCA
# ----------------------------------
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_clust_scaled)
print("Varianza explicada en cada componente PCA:", pca.explained_variance_ratio_)

# Imprimir la suma de varianza
print("Varianza total por PCA:", sum(pca.explained_variance_ratio_))
```

```
Varianza explicada en cada componente PCA: [0.53053033 0.28453161]
Varianza total por PCA: 0.8150619426027885
```

```
# ----------------------
# Lasso Regression (L1)
# ----------------------
lasso = Lasso(alpha=0.1)
lasso.fit(X_train_reg, y_train_reg)
y_pred_lasso = lasso.predict(X_test_reg)
print("Lasso Regression:")
print(f"MAE: {mean_absolute_error(y_test_reg, y_pred_lasso):.2f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test_reg, y_pred_lasso)):.2f}")
print(f"R²: {r2_score(y_test_reg, y_pred_lasso):.4f}")
```

```
Lasso Regression:
MAE: 925179.58
RMSE: 1120386.20
R²: 0.9999
```

```
# ----------------------------------------
# Análisis de Clustering (No Supervisado)
# ----------------------------------------
# Seleccionar características para clustering
features_clust = ['Recaudación', 'Espectadores', 'Pantallas',
                  'Gasto medio/espectador', 'Frecuencia', 'Nº Películas']
X_clust = df[features_clust]

# Aplicar la normalización
scaler = StandardScaler()
X_clust_scaled = scaler.fit_transform(X_clust)

# Convertir el resultado a DataFrame para visualizar
df_scaled = pd.DataFrame(X_clust_scaled, columns=features_clust)

print("Matriz escalada (StandardScaler):")
df_scaled.head()
```

```
# ---------------------
# Clustering: K-Means
# ---------------------

# Configurar hiperparámetros
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10, max_iter=300)
clusters_kmeans = kmeans.fit_predict(X_clust_scaled)

# Métricas de evaluación
print("K-Means Inercia (SSE):", kmeans.inertia_)
try:
    silhouette_kmeans = silhouette_score(X_clust_scaled, clusters_kmeans)
    print("K-Means Silhouette Score:", silhouette_kmeans)
except Exception as e:
    print("No se pudo calcular el Silhouette Score para K-Means:", e)

cluster_labels = {
    0: "Alta recaudación y espectadores",
    1: "Media recaudación y espectadores",
    2: "Baja recaudación y espectadores"
}
df['Cluster_KMeans'] = [cluster_labels[label] for label in clusters_kmeans]

plt.figure(figsize=(16, 5))
sns.scatterplot(
    x=X_pca[:, 0], y=X_pca[:, 1],
    hue=df['Cluster_KMeans'], palette="viridis", s=100, edgecolor='k'
)
plt.title("Clusters con K-Means visualizados en el espacio PCA", fontsize=14)
plt.xlabel("Componente Principal 1", fontsize=12)
plt.ylabel("Componente Principal 2", fontsize=12)
plt.legend(title="Cluster", bbox_to_anchor=(1.00, 1), loc="upper left")
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
K-Means Inercia (SSE): 126.06077452887268
K-Means Silhouette Score: 0.655115161874566
```

```
# ------------------------------
# Pipeline: Arbol de Decisión
# ------------------------------

dt_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor_clf),
    ('classifier', DecisionTreeClassifier(random_state=42))
])
dt_pipeline.fit(X_train_clf, y_train_clf)
y_pred_dt = dt_pipeline.predict(X_test_clf)

print("Decision Tree Classifier:")
print("Accuracy:", accuracy_score(y_test_clf, y_pred_dt))
print("Classification Report:\n", classification_report(y_test_clf, y_pred_dt))

# Visualización de la Matriz de Confusión
plt.figure(figsize=(16, 5))
sns.heatmap(confusion_matrix(y_test_clf, y_pred_dt), annot=True, fmt="d",
            cmap="Greens", cbar=False)
plt.title("Matriz de Confusión - Árbol de Decisión", fontsize=14)
plt.xlabel("Predicción", fontsize=12)
plt.ylabel("Actual", fontsize=12)
plt.tight_layout()
plt.show()

# Validación cruzada para Árbol de Decisión
cv_scores_dt = cross_val_score(dt_pipeline, X_clf, y_clf, cv=5, scoring='accuracy')
mean_accuracy_dt = np.mean(cv_scores_dt)
std_accuracy_dt = np.std(cv_scores_dt)
print("Decision Tree Classifier - Precisión promedio (Validación cruzada):",
      mean_accuracy_dt)
print(f"Desviación estándar de precisión: {std_accuracy_dt:.4f}")
```

```
Decision Tree Classifier:
Accuracy: 1.0
Alta recaudación y espectadores      1.00    1.00    1.00       4
 Baja recaudación y espectadores     1.00    1.00    1.00       4
Media recaudación y espectadores     1.00    1.00    1.00      10
```

```
# ------------------------------------
# Pipeline: Support Vector Machine SVM
# ------------------------------------

svm_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor_clf),
    ('classifier', SVC(kernel='rbf', random_state=42, C=1.0, gamma='scale'))
])
svm_pipeline.fit(X_train_clf, y_train_clf)
y_pred_svm = svm_pipeline.predict(X_test_clf)

print("SVM Classifier Accuracy:", accuracy_score(y_test_clf, y_pred_svm))
print("Classification Report:", classification_report(y_test_clf, y_pred_svm))

# Visualización de la matriz de confusión
plt.figure(figsize=(16, 5))
sns.heatmap(confusion_matrix(y_test_clf, y_pred_svm), annot=True, fmt="d", cmap="Blues", cbar=Fals
plt.title("Matriz de Confusión - SVM", fontsize=14)
plt.xlabel("Predicción", fontsize=12)
plt.ylabel("Actual", fontsize=12)
plt.tight_layout()
plt.show()

# Validación cruzada para SVM
cv_scores_svm = cross_val_score(svm_pipeline, X_clf, y_clf, cv=5, scoring='accuracy')
mean_accuracy_svm = np.mean(cv_scores_svm)
std_accuracy_svm = np.std(cv_scores_svm)
print("SVM Classifier - Precisión promedio (Validación cruzada):", mean_accuracy_svm)
print(f"Desviación estándar de precisión: {std_accuracy_svm:.4f}")
```

```
# ---------------
# Guardar joblib
# ---------------
# Guardamos el modelo de Random Forest para regresión
joblib.dump(rf_model, 'src/models/final_rf_model.pkl')
print("Modelo Random Forest Regression exportado en 'src/models/final_rf_model.pkl'")

# Guardamos el pipeline completo del Árbol de Decisión para clasificación
joblib.dump(dt_pipeline, 'src/models/final_decision_tree_classifier.pkl')
print("Modelo Decision Tree Classifier exportado en 'src/models/final_decision_tree_classifier.pkl'")
```

```
Modelo Random Forest Regression exportado en 'src/models/final_rf_model.pkl'
Modelo Decision Tree Classifier exportado en 'src/models/final_decision_tree_classifier.pkl'
```