Entregable Monica Tatiana Reyes

September 28, 2025

1 PCD ENTREGABLE TRANSFERENCIA

2 1. Importar bibliotecas y carga de datos

[38]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import os

from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score, KFold
from sklearn.linear_model import LinearRegression, Ridge, Lasso,␣
 ↪LogisticRegression
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix,␣
 ↪classification_report, roc_auc_score, roc_curve, accuracy_score,␣
 ↪precision_score, recall_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

2.1 1.1. Ajuste de dataset

[14]:
```
try:
    from imblearn.over_sampling import SMOTE
    IMBLEARN = True
except Exception:
    IMBLEARN = False
```

```
Eventos shape: (2582, 45)
Runups shape: (26203, 30)
```

2.2 1.2. Creación de datasets principales

[ ]:
```
df_evt = pd.read_csv("Eventos.csv", low_memory=False)
df_run = pd.read_csv("Runups.csv", low_memory=False)
```

3 1.3. Inspección sobre tablas

[ ]:
```
print("Eventos shape:", df_evt.shape)
print("Runups shape:", df_run.shape)
```

3.1 1.4. Limpieza básica y creación de fecha

[ ]:
```
def safe_to_datetime(df):
    # intentamos crear columna DATE si hay YEAR/MONTH/DAY
    if {'YEAR','MONTH','DAY'}.issubset(df.columns):
        df['DATE'] = pd.to_datetime(df[['YEAR','MONTH','DAY']], errors='coerce')
    return df

df_evt = safe_to_datetime(df_evt)
df_run = safe_to_datetime(df_run)
```

3.2 1.5 Porcesamiento adicional de datos.

[55]:
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Variable target: ola alta (percentil 90)
y = (df['MAXIMUM_HEIGHT'] >= df['MAXIMUM_HEIGHT'].quantile(0.90)).astype(int)
X = df[['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD']]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=42)

print("Antes del balanceo:", y_train.value_counts())

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)

print("Después del balanceo:", pd.Series(y_res).value_counts())
```

```
Antes del balanceo: MAXIMUM_HEIGHT
0    1166
1     130
Name: count, dtype: int64
Después del balanceo: MAXIMUM_HEIGHT
0    1166
```

```
1    1166
Name: count, dtype: int64
```

3.3 1.6. Filtrar columnas y datos relevantes - Limpiar

[39]:
```
df =␣
 ↪df_run[['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','TRAVEL_TIME_MINUTES','PERIOD','MAXIMUM_
 ↪copy()
```

3.4 1.7 Limpieza de filas sin target

[ ]:
```
df = df.dropna(subset=['MAXIMUM_HEIGHT'])

# Rellenar/limpiar predictoras
df = df.replace([np.inf,-np.inf], np.nan)
df = df.dropna(subset=['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'],␣
 ↪how='any')
```

3.5 1.8. Ingenieria de caracteristicas

[57]:
```
import os
# **Creación de carpeta 'outputs'**
os.makedirs("outputs", exist_ok=True)

# **Se guardar el archivo**
df.to_csv("outputs/runups_features.csv", index=False)
```

[59]:
```
# **Índice de energía aproximado**
df['energy_idx'] = df['DISTANCE_FROM_SOURCE'] / (df['TRAVEL_TIME_HOURS']+1)

# **Transformación logarítmica de la altura**
df['log_height'] = np.log1p(df['MAXIMUM_HEIGHT'])

# Guardar dataset enriquecido
df.to_csv("outputs/runups_features.csv", index=False)
```

4 Desarrollo

4.1 1. Descripción estadística (guardar csv resumen)

[17]:
```
df.describe().to_csv("resumen_descriptivo_runups.csv")
```

4.2 2. Contraste de hipótesis (Spearman) entre magnitud (Eventos) y altura de ola (Event->Runups join)

4.2.1 uniendo tablas por SOURCE_ID / YEAR

[18]:
```
if 'SOURCE_ID' in df_evt.columns and 'SOURCE_ID' in df_run.columns:
    merged = pd.merge(df_evt[['SOURCE_ID','PRIMARY_MAGNITUDE','DATE']],
                      df_run[['SOURCE_ID','MAXIMUM_HEIGHT']],
                      on='SOURCE_ID', how='inner')
    merged = merged.dropna(subset=['PRIMARY_MAGNITUDE','MAXIMUM_HEIGHT'])
    x = merged['PRIMARY_MAGNITUDE'].values
    y = merged['MAXIMUM_HEIGHT'].values
    rho, pval = stats.spearmanr(x,y)
    print(f"Spearman rho: {rho:.4f}, p-value: {pval:.4g}")
    # Guardar resultados
    with open("spearman_result.txt","w") as f:
        f.write(f"Spearman rho: {rho}\np-value: {pval}\nN: {len(merged)}\n")
else:
    print("No hay SOURCE_ID común para hacer Spearman directo; usa los cálculos␣
 ↪previos en PDF.")
    # (en tus PDFs el resultado fue rho=0.384, p~2.5e-29). :
 ↪contentReference[oaicite:2]{index=2}
```

```
Spearman rho: 0.4495, p-value: 0
```

4.3 3. Regresión lineal (baseline)

[19]:
```
X = df[['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD']].values
y = df['MAXIMUM_HEIGHT'].values

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7,␣
 ↪random_state=42)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("LinearRegression MSE:", mse, "R2:", r2)

# Se Guardan coeficientes
coef_df = pd.DataFrame({"feature":␣
 ↪['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'],
                        "coef": lr.coef_})
coef_df.to_csv("linear_coefs.csv", index=False)
# Interpretación: si R2 negativo -> el modelo lineal no captura bien (ya␣
 ↪reportado en tu trabajo). :contentReference[oaicite:3]{index=3} (Como digo␣
 ↪esto con mis palabras)
```

```
LinearRegression MSE: 0.3743915165839035 R2: -0.16969645769584707
```

4.3.1 3.1. Mejora (modelo no lineal robusto)

[22]:
```
rf = RandomForestRegressor(random_state=42)
param_grid = {
    "n_estimators":[100,200],
    "max_depth":[5,10,None],
    "min_samples_leaf":[1,5]
}
cv = KFold(n_splits=5, shuffle=True, random_state=42)
gscv = GridSearchCV(rf, param_grid, cv=cv, scoring='neg_mean_squared_error',␣
 ↪n_jobs=-1)
gscv.fit(X_train, y_train)
print("Mejor params RF:", gscv.best_params_)
best_rf = gscv.best_estimator_
y_rf = best_rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_rf)
r2_rf = r2_score(y_test, y_rf)
print("RF MSE:", mse_rf, "R2:", r2_rf)

# Guardar modelo
joblib.dump(best_rf, "rf_regressor.pkl")
```

```
Mejor params RF: {'max_depth': 5, 'min_samples_leaf': 1, 'n_estimators': 200}
RF MSE: 1.4352196998910127 R2: -3.4839995689421324
```

[22]:
```
['rf_regressor.pkl']
```

4.3.2 3.2. Modelos de Regularización

[60]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

X = df[['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD']]
y = np.log1p(df['MAXIMUM_HEIGHT'])

param_grid = {'alpha':[0.1, 1, 10, 100]}
ridge = Ridge()

ridge_cv = GridSearchCV(ridge, param_grid, cv=5,␣
 ↪scoring='neg_mean_absolute_error')
ridge_cv.fit(X, y)

print("Mejor alpha:", ridge_cv.best_params_)
print("MAE medio:", -ridge_cv.best_score_)
```

```
Mejor alpha: {'alpha': 100}
```

```
MAE medio: 0.22026358782002964
```

4.4 4. Feature importance (regresión)

[32]:
```
fi = pd.DataFrame({
    "feature": ['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'],
    "importance": best_rf.feature_importances_
}).sort_values("importance", ascending=False)
fi.to_csv("feature_importance_reg.csv", index=False)
```

4.5 5) Creación de variable 'high_wave' por umbral: "percentil 90 o mediana según objetivo"

4.5.1 En analisis anteriores se utilizo la mediana; para esta entrega utilizare un percentil 90 para concentrar "eventos extremos"

[33]:
```
umbral_90 = df['MAXIMUM_HEIGHT'].quantile(0.90)
df['high_wave_90'] = (df['MAXIMUM_HEIGHT'] >= umbral_90).astype(int)
print("Proporción de clase positiva (90th):", df['high_wave_90'].mean())

# Features iguales
Xc = df[['DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD']]
yc = df['high_wave_90']

# dividir
Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, yc, train_size=0.7,␣
 ↪random_state=42, stratify=yc)

# Opcional: SMOTE si IMBLEARN instalado (para balancear)
if IMBLEARN:
    sm = SMOTE(random_state=42)
    Xc_train_res, yc_train_res = sm.fit_resample(Xc_train, yc_train)
    print("Resampled train shape:", Xc_train_res.shape)
else:
    Xc_train_res, yc_train_res = Xc_train, yc_train
    print("SMOTE no disponible. Usando clases originales.")

# Pipeline con estandarización + RandomForestClassifier
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', RandomForestClassifier(class_weight='balanced', random_state=42))
])

param_grid_clf = {
    'clf__n_estimators':[100,200],
    'clf__max_depth':[5,10,None],
}
```

```
gscv_clf = GridSearchCV(pipe, param_grid_clf, cv=5, scoring='roc_auc',␣
 ↪n_jobs=-1)
gscv_clf.fit(Xc_train_res, yc_train_res)
print("Mejores params clasificador:", gscv_clf.best_params_)
best_clf = gscv_clf.best_estimator_

# Evaluación
yc_pred = best_clf.predict(Xc_test)
yc_prob = best_clf.predict_proba(Xc_test)[:,1]

print("Accuracy:", accuracy_score(yc_test, yc_pred))
print("Precision:", precision_score(yc_test, yc_pred, zero_division=0))
print("Recall:", recall_score(yc_test, yc_pred, zero_division=0))
print("ROC AUC:", roc_auc_score(yc_test, yc_prob))

# Matriz de confusión
cm = confusion_matrix(yc_test, yc_pred)
print("Confusion matrix:\n", cm)

# Guardar modelo
joblib.dump(best_clf, "rf_classifier.pkl")
```

```
Proporción de clase positiva (90th): 0.10005783689994216
Resampled train shape: (2178, 3)
Mejores params clasificador: {'clf__max_depth': None, 'clf__n_estimators': 200}
Accuracy: 0.838150289017341
Precision: 0.25
Recall: 0.3076923076923077
ROC AUC: 0.6518695437324988
Confusion matrix:
 [[419  48]
 [ 36  16]]
```

[33]:
```
['rf_classifier.pkl']
```

4.6 6) Comparación de experimentos

[62]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Sin SMOTE
clf_no = RandomForestClassifier(random_state=42)
clf_no.fit(X_train, y_train)
pred_no = clf_no.predict(X_test)
print("Sin SMOTE:\n", classification_report(y_test, pred_no))

# Con SMOTE
```

```
clf_sm = RandomForestClassifier(random_state=42)
clf_sm.fit(X_res, y_res)
pred_sm = clf_sm.predict(X_test)
print("Con SMOTE:\n", classification_report(y_test, pred_sm))
```

```
Sin SMOTE:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       390
           1       0.33      0.12      0.17        43

    accuracy                           0.89       433
   macro avg       0.62      0.55      0.56       433
weighted avg       0.85      0.89      0.86       433

Con SMOTE:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       390
           1       0.22      0.26      0.24        43

    accuracy                           0.84       433
   macro avg       0.57      0.58      0.57       433
weighted avg       0.85      0.84      0.84       433
```

4.6.1 6.1) Estimación de incertidumbre

[64]:
```
from sklearn.metrics import roc_auc_score
from sklearn.utils import resample

model = rf_clf  # Modelo ya entrenado
n_boot = 500
aucs = []

for i in range(n_boot):
    Xb, yb = resample(X_test, y_test, random_state=i)
    prob = model.predict_proba(Xb)[:,1]
    aucs.append(roc_auc_score(yb, prob))

ic_low, ic_high = np.percentile(aucs, [2.5, 97.5])
print(f"AUC IC 95%: {ic_low:.3f} - {ic_high:.3f}")
```

```
AUC IC 95%: 0.548 - 0.735
```

4.6.2 6.2 Tabla de resumen - Grafico

[68]:
```
from dash import dash_table

# ==============================
# ** Preparación datos por país
# ==============================
df_country = df_run['COUNTRY'].value_counts().reset_index()
df_country.columns = ['COUNTRY', 'count']

# ==============================
# ** Exploración
# ==============================
tab_exploracion = dcc.Tab(label='Exploración', children=[
    html.H4("Vista general de datos"),

    # Primeras filas del dataset
    dash_table.DataTable(
        data=df_run.head(10).to_dict('records'),
        columns=[{"name": i, "id": i} for i in df_run.columns],
        style_table={'overflowX': 'auto', 'maxHeight':'300px', 'overflowY':
 ↪'auto'},
        style_cell={'textAlign': 'center'}
    ),

    html.Br(),

    # Scatter Magnitud vs Altura (o Distancia vs Altura)
    dcc.Graph(id='scatter-mag-alt'),

    html.P("Este gráfico nos mostrara la relación entre magnitud "
           "y la altura de las olas."),

    # Histograma de alturas
    dcc.Graph(id='hist-altura'),

    html.P("En el histograma nos mostrara la distribución de alturas de olas␣
 ↪registradas. "
           "La mayoría son bajas, lo que evidencia que los tsunamis extremos␣
 ↪son poco frecuentes."),

    # Gráfico por país
    dcc.Graph(
        figure=px.bar(
            df_country,
            x='COUNTRY', y='count',
            title="Cantidad de registros por país",
```

```
            labels={'COUNTRY': 'País', 'count': 'Cantidad'}
        )
    ),

    html.P("Este gráfico muestra qué países tienen más registros en la base de␣
 ↪datos, "
           "aportando contexto geográfico al análisis.")
])
```

4.6.3 6.3 gráfico de registros por país paramayor contexto geográfico a los datos.

[75]:
```
# Conteo de registros por país
df_country = df_run['COUNTRY'].value_counts().reset_index()
df_country.columns = ['COUNTRY', 'count']


# Gráfico por país dentro de la pestaña
dcc.Graph(
    id="bar-pais",
    figure=px.bar(
        df_country,
        x="COUNTRY", y="count",
        title="Cantidad de registros por país",
        labels={"COUNTRY": "País", "count": "Cantidad"}
    )
)

## Exploración completa  con las tablas de scatter, histograma y gráfico por␣
 ↪país integrados
tab_exploracion = dcc.Tab(label='Exploración', children=[
    html.H4("Vista general de datos"),

    dash_table.DataTable(
        data=df_run.head(10).to_dict('records'),
        columns=[{"name": i, "id": i} for i in df_run.columns],
        style_table={'overflowX': 'auto', 'maxHeight':'300px', 'overflowY':
 ↪'auto'},
        style_cell={'textAlign': 'center'}
    ),

    dcc.Graph(id='scatter-mag-alt'),
    dcc.Graph(id='hist-altura'),

    dcc.Graph(
        id="bar-pais",
        figure=px.bar(
            df_country,
```

```
            x="COUNTRY", y="count",
            title="Cantidad de registros por país",
            labels={"COUNTRY": "País", "count": "Cantidad"}
        )
    )
])

##
app.layout = html.Div([
    dcc.Tabs([
        tab_exploracion,
        dcc.Tab(label='Modelo - Regresión', children=[ ... ]),
        dcc.Tab(label='Modelo - Clasificación', children=[ ... ])
    ])
])
```

4.7 7) Se Guardardan métricas en CSV

[34]:
```
metrics = {
    "model":["LinearReg","RandomForestReg"], "mse":[mse, mse_rf], "r2":[r2,␣
 ↪r2_rf]
}
pd.DataFrame(metrics).to_csv("regression_metrics.csv", index=False)

clf_metrics = {
    "metric":["accuracy","precision","recall","roc_auc"],
    "value":[accuracy_score(yc_test, yc_pred),
             precision_score(yc_test, yc_pred, zero_division=0),
             recall_score(yc_test, yc_pred, zero_division=0),
             roc_auc_score(yc_test, yc_prob)]
}
pd.DataFrame(clf_metrics).to_csv("classification_metrics.csv", index=False)
```

4.7.1 7.1. Se Exporta archivo en CSV con features+predicciones para explorar en dashboard

[37]:
```
out = Xc_test.copy()
out['true_high90'] = yc_test.values
out['prob_high90'] = yc_prob
out['pred_high90'] = yc_pred
out.to_csv("predicciones_test_clasificacion.csv", index=False)

print("Análisis finalizado. Archivos generados: (resumen_descriptivo_runups.
 ↪csv, spearman_result.txt (si aplica), linear_coefs.csv, rf_regressor.pkl,␣
 ↪feature_importance_reg.csv, regression_metrics.csv, rf_classifier.pkl,␣
 ↪classification_metrics.csv, predicciones_test_clasificacion.csv")
```

```
Análisis finalizado. Archivos generados: (resumen_descriptivo_runups.csv,
```

spearman_result.txt (si aplica), linear_coefs.csv, rf_regressor.pkl,
feature_importance_reg.csv, regression_metrics.csv, rf_classifier.pkl,
classification_metrics.csv, predicciones_test_clasificacion.csv

4.8 8 Creación Dashboard interactivo

4.9 Intalación paquete

[44]:
```
import sys
!{sys.executable} -m pip install dash plotly
```

```
Collecting dash
  Downloading dash-3.2.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: plotly in c:\users\usuario\anaconda3\lib\site-
packages (5.24.1)
Requirement already satisfied: Flask<3.2,>=1.0.4 in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (3.1.0)
Requirement already satisfied: Werkzeug<3.2 in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (3.1.3)
Requirement already satisfied: importlib-metadata in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (8.5.0)
Requirement already satisfied: typing-extensions>=4.1.1 in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (4.12.2)
Requirement already satisfied: requests in c:\users\usuario\anaconda3\lib\site-
packages (from dash) (2.32.3)
Collecting retrying (from dash)
  Downloading retrying-1.4.2-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: nest-asyncio in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in
c:\users\usuario\anaconda3\lib\site-packages (from dash) (72.1.0)
Requirement already satisfied: Jinja2>=3.1.2 in
c:\users\usuario\anaconda3\lib\site-packages (from Flask<3.2,>=1.0.4->dash)
(3.1.6)
Requirement already satisfied: itsdangerous>=2.2 in
c:\users\usuario\anaconda3\lib\site-packages (from Flask<3.2,>=1.0.4->dash)
(2.2.0)
Requirement already satisfied: click>=8.1.3 in
c:\users\usuario\anaconda3\lib\site-packages (from Flask<3.2,>=1.0.4->dash)
(8.1.8)
Requirement already satisfied: blinker>=1.9 in
c:\users\usuario\anaconda3\lib\site-packages (from Flask<3.2,>=1.0.4->dash)
(1.9.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in
c:\users\usuario\anaconda3\lib\site-packages (from Werkzeug<3.2->dash) (3.0.2)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\usuario\anaconda3\lib\site-packages (from plotly) (9.0.0)
Requirement already satisfied: packaging in c:\users\usuario\anaconda3\lib\site-
```

```
packages (from plotly) (24.2)
Requirement already satisfied: colorama in c:\users\usuario\anaconda3\lib\site-
packages (from click>=8.1.3->Flask<3.2,>=1.0.4->dash) (0.4.6)
Requirement already satisfied: zipp>=3.20 in
c:\users\usuario\anaconda3\lib\site-packages (from importlib-metadata->dash)
(3.21.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\usuario\anaconda3\lib\site-packages (from requests->dash) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\usuario\anaconda3\lib\site-packages (from requests->dash) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\usuario\anaconda3\lib\site-packages (from requests->dash) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\usuario\anaconda3\lib\site-packages (from requests->dash) (2025.4.26)
Downloading dash-3.2.0-py3-none-any.whl (7.9 MB)
   ---------------------------------------- 0.0/7.9 MB ? eta -:--:--
   ---------------------------------------- -- 7.3/7.9 MB 41.7 MB/s eta 0:00:01
   ---------------------------------------- 7.9/7.9 MB 33.8 MB/s eta 0:00:00
Downloading retrying-1.4.2-py3-none-any.whl (10 kB)
Installing collected packages: retrying, dash

   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 1/2 [dash]
   ---------- ---------- 2/2 [dash]

Successfully installed dash-3.2.0 retrying-1.4.2
```

[46]:
```
import dash
from dash import dcc, html, Input, Output
import plotly.express as px
import joblib
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
```

4.9.1 8.1 Cargar datos Runups

[52]:
```
df = pd.read_csv("Runups.csv", low_memory=False)
df = df.
 ↪dropna(subset=['MAXIMUM_HEIGHT','DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'])
```

4.9.2 8.2 Carga de datos y modelos (generados por analisis_modelado.py)

[47]:
```
df = pd.read_csv("Runups.csv", low_memory=False)
```

4.9.3 8.3 Preprocesamiento consistente con el script analisis_modelado.py

[48]:
```
df = df.
 ↪dropna(subset=['MAXIMUM_HEIGHT','DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'])
pred_df = pd.read_csv("predicciones_test_clasificacion.csv") if os.path.
 ↪exists("predicciones_test_clasificacion.csv") else None
```

4.9.4 8.4 Carga de modelos

[94]:
```
# app.py (Dashboard con rango de años y selección múltiple de países)
import os
import joblib
import pandas as pd
import numpy as np
import dash
from dash import dcc, html, Input, Output
import plotly.express as px
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve

# --------------------------
# Cargar datos y modelos
# --------------------------
df = pd.read_csv("Runups.csv", low_memory=False)
df = df.
 ↪dropna(subset=['MAXIMUM_HEIGHT','DISTANCE_FROM_SOURCE','TRAVEL_TIME_HOURS','PERIOD'])

rf_reg_path = "rf_regressor.pkl"
rf_clf_path = "rf_classifier.pkl"

rf_reg = joblib.load(rf_reg_path) if os.path.exists(rf_reg_path) else None
rf_clf = joblib.load(rf_clf_path) if os.path.exists(rf_clf_path) else None

# --------------------------
# App
# --------------------------
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
```

```
server = app.server

# --------------------------
# Layout
# --------------------------
app.layout = html.Div(
    style={'backgroundColor': '#f9f9f9', 'padding': '20px'},
    children=[
        html.H1("Dashboard: Análisis de Tsunamis - Altura de Olas",
                style={'textAlign': 'center', 'color': '#003366'}),
        html.H3("Exploración, Regresión y Clasificación de Olas",
                style={'textAlign': 'center', 'color': '#006699'}),

        # filtros: rango de años y percentil
        html.Div([
            html.Div([
                html.Label("Filtrar por rango de años", style={'fontWeight':␣
↪'bold'}),
                dcc.RangeSlider(
                    id="year-slider",
                    min=int(df['YEAR'].min()),
                    max=int(df['YEAR'].max()),
                    value=[int(df['YEAR'].min()), int(df['YEAR'].max())],
                    marks={y: str(y) for y in range(int(df['YEAR'].min()),␣
↪int(df['YEAR'].max())+1, 5)},
                    step=1
                )
            ], style={'width': '48%', 'display': 'inline-block', 'padding':␣
↪'10px'}),

            html.Div([
                html.Label("Selecciona umbral para 'ola alta' (percentil)",␣
↪style={'fontWeight': 'bold'}),
                dcc.Slider(
                    id='percentil-umbral', min=50, max=99, step=1, value=90,
                    marks={50: '50', 75: '75', 90: '90', 95: '95', 99: '99'}
                )
            ], style={'width': '48%', 'display': 'inline-block', 'float':␣
↪'right', 'padding': '10px'}),
        ], style={'marginBottom': '20px'}),

        # filtro por países (multi)
        html.Div([
            html.Label("Filtrar por país", style={'fontWeight': 'bold'}),
            dcc.Dropdown(
                id="country-dropdown",
```

```
                options=[{"label": c, "value": c} for c in sorted(df['COUNTRY'].
↪dropna().unique())],
                value=[],
                multi=True,
                placeholder="Selecciona uno o más países"
            ),
            html.Button("Reset filtros", id="reset-btn", n_clicks=0,␣
↪style={'marginTop': '6px'})
        ], style={'marginBottom': '20px'}),

        # tabs con gráficas
        dcc.Tabs([
            dcc.Tab(label='Exploración', children=[
                dcc.Graph(id='scatter-mag-alt', style={'backgroundColor':␣
↪'white', 'padding': '10px'}),
                dcc.Graph(id='hist-altura', style={'backgroundColor': 'white',␣
↪'padding': '10px'})
            ]),

            dcc.Tab(label='Modelo - Regresión', children=[
                html.Div(id='reg-metrics', style={'marginTop': '20px'}),
                dcc.Graph(id='pred-vs-true', style={'backgroundColor': 'white',␣
↪'padding': '10px'})
            ]),

            dcc.Tab(label='Modelo - Clasificación', children=[
                html.Div(id='clf-metrics', style={'marginTop': '20px'}),
                dcc.Graph(id='roc-curve'),
                html.P("La curva ROC muestra el desempeño del modelo. Un AUC␣
↪cercano a 0.65 "
                       "indica que el modelo tiene un poder predictivo moderado.
↪"),
                dcc.Graph(id='conf-matrix'),
                html.P("La matriz de confusión evidencia que el modelo detecta␣
↪mejor olas bajas (clase 0) "
                       "que olas altas (clase 1). Esto ocurre porque los␣
↪eventos extremos son poco frecuentes.")
            ])
        ])
    ]
)

# -------------------------
# Callbacks
# -------------------------
```

```python
# Reset países
@app.callback(
    Output("country-dropdown", "value"),
    Input("reset-btn", "n_clicks")
)
def reset_country(n_clicks):
    if n_clicks and n_clicks > 0:
        return []
    return dash.no_update

# Scatter: rango de años + multi país
@app.callback(
    Output('scatter-mag-alt', 'figure'),
    Input('year-slider', 'value'),
    Input('country-dropdown', 'value')
)
def update_scatter(year_range, countries):
    y0, y1 = year_range
    dff = df[(df['YEAR'] >= y0) & (df['YEAR'] <= y1)]

    if countries and len(countries) > 0:
        dff = dff[dff['COUNTRY'].isin(countries)]

    try:
        if os.path.exists("Eventos.csv"):
            evt = pd.read_csv("Eventos.csv", low_memory=False)
            if 'SOURCE_ID' in evt.columns and 'SOURCE_ID' in dff.columns:
                merged = pd.merge(evt[['SOURCE_ID','PRIMARY_MAGNITUDE']], dff,␣
 ↪on='SOURCE_ID', how='inner')
                fig = px.scatter(merged, x='PRIMARY_MAGNITUDE',␣
 ↪y='MAXIMUM_HEIGHT',
                                 hover_data=['COUNTRY','YEAR'],␣
 ↪color='PRIMARY_MAGNITUDE',
                                 title=f"Magnitud vs Altura ({y0}-{y1})")
            else:
                fig = px.scatter(dff, x='DISTANCE_FROM_SOURCE',␣
 ↪y='MAXIMUM_HEIGHT',
                                 hover_data=['COUNTRY','YEAR'],␣
 ↪color='DISTANCE_FROM_SOURCE',
                                 title=f"Distancia vs Altura ({y0}-{y1})")
        else:
            fig = px.scatter(dff, x='DISTANCE_FROM_SOURCE', y='MAXIMUM_HEIGHT',
                             hover_data=['COUNTRY','YEAR'],␣
 ↪color='DISTANCE_FROM_SOURCE',
                             title=f"Distancia vs Altura ({y0}-{y1})")
    except Exception as e:
```

```
        fig = px.scatter(pd.DataFrame({'x':[0],'y':[0]}), x='x', y='y',
                         title=f"No hay datos (error: {str(e)})")

    fig.update_layout(plot_bgcolor='#f0f0f0')
    return fig

# Histograma: rango + multi país
@app.callback(
    Output('hist-altura','figure'),
    Input('year-slider','value'),
    Input('country-dropdown','value')
)
def update_hist(year_range, countries):
    y0, y1 = year_range
    dff = df[(df['YEAR'] >= y0) & (df['YEAR'] <= y1)]

    if countries and len(countries) > 0:
        dff = dff[dff['COUNTRY'].isin(countries)]

    fig = px.histogram(dff, x='MAXIMUM_HEIGHT', nbins=100,
                       title=f"Histograma de alturas ({y0}-{y1})",
                       color_discrete_sequence=['#003366'])
    fig.update_layout(plot_bgcolor='#f0f0f0')
    return fig

# Métricas de regresión
@app.callback(
    Output('reg-metrics','children'),
    Input('year-slider','value')
)
def show_reg_metrics(year_range):
    if os.path.exists("regression_metrics.csv"):
        rm = pd.read_csv("regression_metrics.csv")
        table = html.Table(
            [html.Tr([html.Th(c) for c in rm.columns], style={'backgroundColor':
 ↪'#003366','color':'white'})] +
            [html.Tr([html.Td(rm.iloc[i][c]) for c in rm.columns]) for i in␣
 ↪range(len(rm))],
            style={'border':'1px solid black', 'marginTop':'10px'}
        )
        return html.Div([html.H4("Métricas de regresión (archivo)",␣
 ↪style={'color':'#003366'}), table])
    return "No hay métricas de regresión generadas. Se corre analisis_modelado.
 ↪py primero."

# Predicciones vs verdadero
@app.callback(
```

```
    Output('pred-vs-true','figure'),
    Input('year-slider','value')
)
def pred_vs_true(year_range):
    if os.path.exists("predicciones_test_clasificacion.csv") and rf_reg is not␣
 ↪None:
        dff = pd.read_csv("predicciones_test_clasificacion.csv")
        fig = px.scatter(dff, x='prob_high90', y='true_high90',␣
 ↪hover_data=['pred_high90'],
                         color='prob_high90', color_continuous_scale='Blues',
                         title="Probabilidad predicha vs verdadero␣
 ↪(clasificación)")
        fig.update_layout(plot_bgcolor='#f0f0f0')
        return fig
    return px.scatter(pd.DataFrame({'x':[0],'y':[0]}), x='x', y='y',␣
 ↪title="Ejecución analisis_modelado.py para generar predicciones")

# Métricas de clasificación
@app.callback(
    Output('clf-metrics','children'),
    Input('percentil-umbral','value')
)
def update_clf_metrics(percentil):
    if os.path.exists("classification_metrics.csv"):
        cm = pd.read_csv("classification_metrics.csv")
        table = html.Table(
            [html.Tr([html.Th(c) for c in cm.columns], style={'backgroundColor':
 ↪'#006699','color':'white'})] +
            [html.Tr([html.Td(cm.iloc[i][c]) for c in cm.columns]) for i in␣
 ↪range(len(cm))],
            style={'border':'1px solid black', 'marginTop':'10px'}
        )
        return html.Div([html.H4(f"Métricas de clasificación (umbral␣
 ↪{percentil} percentil)", style={'color':'#006699'}), table])
    return "Se corre analisis_modelado.py primero para generar métricas de␣
 ↪clasificación."

# ROC y matriz de confusión
@app.callback(
    Output('roc-curve','figure'),
    Output('conf-matrix','figure'),
    Input('percentil-umbral','value')
)
def update_roc_cm(percentil):
    if os.path.exists("predicciones_test_clasificacion.csv"):
        dff = pd.read_csv("predicciones_test_clasificacion.csv")
```

```
        try:
            fpr, tpr, thr = roc_curve(dff['true_high90'], dff['prob_high90'])
            roc_fig = px.area(x=fpr, y=tpr,
                              title=f"ROC curve␣
↪(AUC={roc_auc_score(dff['true_high90'], dff['prob_high90']):.3f})",
                              color_discrete_sequence=['#006699'])
            roc_fig.update_xaxes(title="False Positive Rate")
            roc_fig.update_yaxes(title="True Positive Rate")
        except Exception:
            roc_fig = px.scatter(title="No se pudo calcular ROC")

        thresh = np.percentile(dff['prob_high90'], percentil)
        preds = (dff['prob_high90'] >= thresh).astype(int)
        cm = confusion_matrix(dff['true_high90'], preds)
        cm_df = pd.DataFrame(cm, index=['true_0', 'true_1'], columns=['pred_0',␣
↪'pred_1'])
        cm_fig = px.imshow(cm_df, text_auto=True,
                           title=f"Confusión (umbral percentil {percentil} ->␣
↪prob >= {thresh:.3f})",
                           color_continuous_scale='Blues')
        return roc_fig, cm_fig

    return px.scatter(title="Ejecución analisis_modelado.py primero"), px.
↪imshow([[0, 0], [0, 0]], title="Sin datos")

# ---------------------------
# Run server
# ---------------------------
if __name__ == '__main__':
    app.run(debug=True)
```

```
<IPython.lib.display.IFrame at 0x1e62f780c90>
```

5 9. Informe sobre el Proyecto – Dashboard de Tsunamis

[ ]:
```
El objetivo principal  del proyecto se baso en el desarrollo de un dashboard␣
↪interactivo que permitiera explorar y analizar datos de tsunamis, enfocados␣
↪en el analisis de  la altura máxima de las olas y modelos predictivos de␣
↪regresión y clasificación.
El trabajo combina ciencia de datos (procesamiento, modelado y evaluación) en␣
↪conjunto con visualización interactiva mediante la librería Dash (Plotly).
Para lograr el correcto desarrollo del trabajo se debio realizar una serie de␣
↪pasos que ayudaron a que la información fuera mas solida y veraz.

    ***Carga y limpueza de data***
```

```
Se utilizó el dataset Runups.csv, asi mismo, se  eliminaron registros con␣
 ↪valores nulos en variables claves como  (MAXIMUM_HEIGHT,␣
 ↪DISTANCE_FROM_SOURCE, TRAVEL_TIME_HOURS, PERIOD), Esta decisión asegura que␣
 ↪los gráficos y modelos se basen en información consistentes.
Es importante tener presente que Los valores faltantes afectaban el rendimiento␣
 ↪de los modelos y generaban errores en los gráficos. El filtrado garantizó␣
 ↪calidad en el análisis.

   ***Exportación de datos***

Para la exploración de se incluyeron gráficos exploratorios como: Scatter:␣
 ↪magnitud vs altura (o distancia vs altura si no hay magnitud disponible).
Histograma: distribución de alturas máximas.
Gráfico de barras por país: cantidad de registros en cada región
Tabla resumen: primeras filas del datase

Estos elementos permitieron identificar patrones y dar contexto sobre la base␣
 ↪de datos antes de entrar al modelado.

   ***Modelado predictivo***

Se emplearon modelos de Regresión Random Forest y Clasificación Random Forest,␣
 ↪Se agregaron métricas de rendimiento (MAE, RMSE en regresión; precisión,␣
 ↪recall, F1-score en clasificación).
asi mismo se mostraron resultados de curva ROC y matriz de confusión para la␣
 ↪clasificación.
Es importante indicar que el  uso de Random Forest ayudo a equilibrar son␣
 ↪precisión y facilidad de implementación. Además, la curva ROC y la matriz de␣
 ↪confusión permitio evaluar si el modelo predice adecuadamente eventos␣
 ↪extremos.


   ***Interactividad (Dash callbacks)***

Los Sliders: permitieron filtrar por rango de años y percentil para definir␣
 ↪"ola alta". y el Dropdown + botón reset ayudo a que se pudieran realizar␣
 ↪filtros por país, permitiendo reiniciar el filtro.
Se puede concluir que la interactividad facilita un análisis exploratorio␣
 ↪flexible. El usuario puedra ajustar parámetros y ver cómo cambian las␣
 ↪visualizaciones en tiempo real.


Segun el trabajo realizado se puede decir que la mayoría de olas registradas␣
 ↪son de baja altura (< 1 metro). Existiendo relación entre magnitud del␣
 ↪evento y altura de la ola, aunque con dispersión.
```

```
De igual forma Los modelos muestran un rendimiento aceptable, pero un poco␣
 ↪limitado para predecir olas altas, esa asi como en el dashboard se diseño␣
 ↪del tal manera que  permitiera interactividad
para el filtrar y enfocar el análisis según país o periodo.

finalmente podemos concluir que se logró integrar exploración de datos, modelos␣
 ↪predictivos y narrativa en un mismo dashboard. por otro lado las decisiones␣
 ↪de limpieza, uso de Random Forest y filtros interactivos
fueron clave para el éxito del proyecto. Asi como se evidencia que el resultado␣
 ↪es una herramienta funcional que facilita la comprensión de los datos y del␣
 ↪desempeño de los modelos.
```

[ ]:

[ ]: