# TEAM CHALLENGE: CONSTRUCCIÓN DE PIPELINES CON SCIKIT-LEARN

# CONTENIDO

# INTRODUCCIÓN Y OBJETIVO

Datos relacionados con las características de los vuelos y la satisfacción de sus clientes.

Determinar si los clientes estuvieron satisfechos o no en sus vuelos

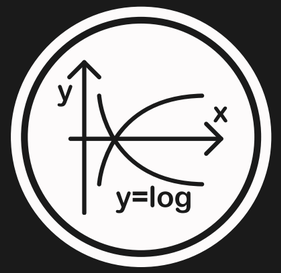
# PIPELINES

**01.**

Limpieza de Nulos y Duplicados

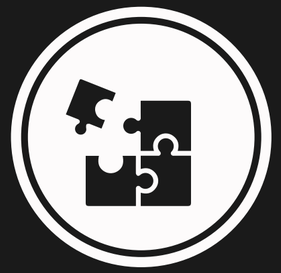
**02.**

Identificación de Tipos de Columnas




**03.**

Aplicar Logaritmo a Variables Numéricas



**04.**

Codificación y Escalado

**05.**

Entrenamiento y Selección del Mejor Modelo

# 01. LIMPIEZA DE NULOS Y DUPLICADOS

```python
# Limpieza de nulos y duplicados
class DropHighNullColumns(BaseEstimator, TransformerMixin):
    def __init__(self, threshold=0.15):
        self.threshold = threshold
        self.cols_to_drop = []

    def fit(self, X, y=None):
        null_percentage = X.isnull().mean()
        self.cols_to_drop = null_percentage[null_percentage > self.threshold].index.tolist()
        return self

    def transform(self, X):
        return X.drop(columns=self.cols_to_drop, errors="ignore")

class DropRemainingNulls(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.dropna()

class DropDuplicates(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.drop_duplicates()

null_pipeline = Pipeline([
    ("drop_high_nulls", DropHighNullColumns(threshold=0.15)),
    ("drop_remaining_nulls", DropRemainingNulls()),
    ("drop_duplicates", DropDuplicates())
])

df_train_cleaned = null_pipeline.fit_transform(df_train)
df_train_cleaned.info()
```

# 02. IDENTIFICACIÓN DE TIPOS DE COLUMNAS

```python
# Identificación de tipos de columnas
class IdentifyColumns(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.numerical_cols = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival Delay in Minutes']
        self.binary_cols = [col for col in X.select_dtypes(include=['object']).columns if X[col].nunique() == 2]
        if 'satisfaction' in self.binary_cols:
            self.binary_cols.remove('satisfaction')
        self.target = 'satisfaction'
        self.onehot_cols = [col for col in X.columns if col not in self.numerical_cols and col not in self.binary_cols and col != self.target]
        self.numerical_cols_log = [col for col in self.numerical_cols if col != 'Age']
        return self

    def transform(self, X):
        return X

identify_columns_pipeline = IdentifyColumns()
identify_columns_pipeline.fit(df_train_cleaned)

numerical_cols_log = identify_columns_pipeline.numerical_cols_log
binary_cols = identify_columns_pipeline.binary_cols
onehot_cols = identify_columns_pipeline.onehot_cols
numerical_cols = identify_columns_pipeline.numerical_cols

print(f'Tipos de columnas identificadas correctamente.')
```

# 03. APLICAR LOGARITMO A VARIABLES NUMÉRICAS

```
# Aplicar Logaritmo a Variables Numéricas
def safe_log_transform(X):
    """Aplica np.log1p() solo a las columnas numéricas seleccionadas."""
    X = X.copy()
    if isinstance(X, pd.DataFrame):
        for col in numerical_cols_log:
            X[col] = pd.to_numeric(X[col], errors="coerce")  # Convertir a float
            X[col] = np.log1p(X[col])  # Aplicar log1p
    else:
        X = np.log1p(X.astype(float))  # Para arrays numpy

    return X

log_pipeline = FunctionTransformer(safe_log_transform)
df_train_cleaned[numerical_cols_log] = log_pipeline.fit_transform(df_train_cleaned[numerical_cols_log])

df_train[numerical_cols].hist(figsize = (5,5))
plt.tight_layout
plt.show()
```

# 04. CODIFICACIÓN Y ESCALADO

```
# One hot encoding y estandarización
def label_encode_binary(X):
    X_encoded = X.copy()
    for col in X_encoded.columns:
        X_encoded[col] = X_encoded[col].astype("category").cat.codes
    return X_encoded

def binarize_satisfaction(X):
    return X.replace({"neutral or dissatisfied": 0, "satisfied": 1}).infer_objects(copy=False)


preprocessor = ColumnTransformer([
    ("target", FunctionTransformer(binarize_satisfaction), ["satisfaction"]),
    ("binary", FunctionTransformer(label_encode_binary), binary_cols),
    ("onehot", Pipeline([
        ("encoder", OneHotEncoder(handle_unknown="ignore", sparse_output=True)),
        ("svd"  TruncatedSVD(n_components=50))  # Reduce dimensiones para evitar MemoryError
    ]), onehot_cols),
    ("scaler", StandardScaler(), numerical_cols)
], remainder='passthrough')

preprocessing_pipeline = Pipeline([
    ("preprocessor", preprocessor)
])

df_train_transformed = preprocessing_pipeline.fit_transform(df_train_cleaned)
print(f'Columnas transformadas y estandarizadas.')
```

# 05. ENTRENAMIENTO Y SELECCIÓN DEL MEJOR MODELO

```
# Train_test_split
train_set, val_set = train_test_split(df_train_transformed, test_size=0.2, random_state=42)
X_train = train_set[:, 1:]
y_train = train_set[:, 0]
X_val = val_set[:, 1:]
y_val = val_set[:, 0]

# Entrenamiento y selección del mejor modelo
models = {
    "RandomForest": RandomForestClassifier(),
    "GradientBoosting": GradientBoostingClassifier(),
    "LogisticRegression": LogisticRegression(),
    "LightGBM": lgb.LGBMClassifier()
}

scores = {}
for model_name, model in models.items():
    pipeline_model = Pipeline([("model", model)])
    pipeline_model.fit(X_train, y_train)
    y_pred = pipeline_model.predict(X_val)
    balanced_acc = balanced_accuracy_score(y_val, y_pred)
    scores[model_name] = balanced_acc
    print(f"{model_name} - Balanced Accuracy: {balanced_acc:.4f}")

best_model_name = max(scores, key=scores.get)
print(f"\nModelo seleccionado para optimización: {best_model_name}")

param_grid = {
    "RandomForest": {"model__n_estimators": [50, 100, 200], "model__max_depth": [None, 10, 20]},
    "GradientBoosting": {"model__n_estimators": [50, 100], "model__learning_rate": [0.01, 0.1]},
    "LightGBM": {"model__num_leaves": [31, 50], "model__learning_rate": [0.01, 0.1]}
}

grid_search = GridSearchCV(Pipeline([("model", models[best_model_name])]), param_grid[best_model_name], cv=5, scoring="balanced_accuracy", n_jobs=-1)
grid_search.fit(X_train, y_train)
```

# CONSTRUCCIÓN DEL PIPELINE FINAL Y GUARDADO DEL MODELO

```python
# Construcción del pipeline completo
full_pipeline = Pipeline([
    ("null_handling", null_pipeline),
    ("column_identification", identify_columns_pipeline),
    ("log_transform", log_pipeline),
    ("preprocessor", preprocessing_pipeline),
    ("model", grid_search.best_estimator_)
])

# Guardar el pipeline completo con dill
BASE_PATH = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Sube un nivel al directorio raíz del proyecto
MODEL_DIR = os.path.join(BASE_PATH, "models")

os.makedirs(MODEL_DIR, exist_ok=True)
model_path = os.path.join(MODEL_DIR, "best_pipeline.pkl")

with open(model_path, "wb") as f:
    dill.dump(full_pipeline, f)

print(f"\nPipeline completo guardado en {model_path}")
```

# APLICACIÓN DE LOS PIPELINES

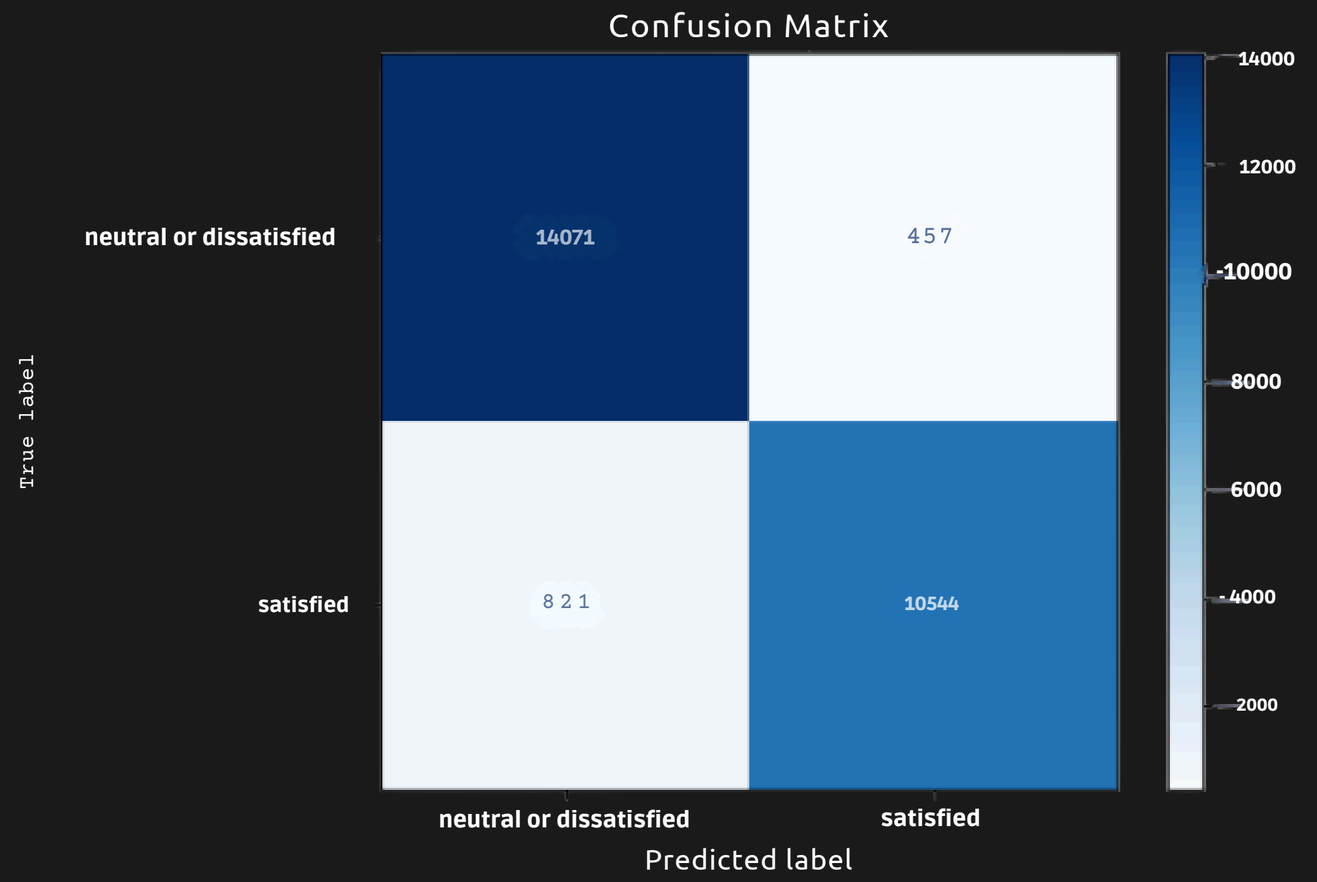
```
# Filtrar FutureWarnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Se asume que 'df_test' ya está cargado y tiene 'id' como índice.
y_test = df_test["satisfaction"].replace({"neutral or dissatisfied": 0, "satisfied": 1}).infer_objects(copy=False)
X_test = df_test.drop(columns=["satisfaction"])

# Añadir la columna dummy 'satisfaction' para que el pipeline procese los datos igual que en entrenamiento
X_test_pipeline = X_test.copy()
X_test_pipeline["satisfaction"] = "neutral or dissatisfied"

# Recuperar la variable 'numerical_cols_log' desde el transformer 'column_identification'
numerical_cols_log = best_pipeline.named_steps["column_identification"].numerical_cols_log

# Obtener los índices de las filas que sobreviven la limpieza (nulos y duplicados)
X_test_clean = best_pipeline.named_steps["null_handling"].transform(X_test_pipeline)
idx_clean = X_test_clean.index

# Transformar hasta el preprocesador y remover la columna del target (posición 0)
X_test_transformed = best_pipeline[:-1].transform(X_test_pipeline)
X_test_final = X_test_transformed[:, 1:]

# Realizar las predicciones con el modelo entrenado
y_pred = best_pipeline.named_steps["model"].predict(X_test_final)

# Alinear y_test con los índices de las filas procesadas
y_test_aligned = y_test.loc[idx_clean]

print(f'¡Los pipelines se han aplicado correctamente!')
```

# RESULTADOS

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.94 | 0.97 | 0.96 | 14528 |
| 1 | 0.96 | 0.93 | 0.94 | 11365 |
| accuracy | | | 0.95 | 25893 |
| macro avg | 0.95 | 0.95 | 0.95 | 25893 |
| weighted avg | 0.95 | 0.95 | 0.95 | 25893 |

# RESULTADOS


Confusion Matrix
neutral or dissatisfied
satisfied
True label
14071
457
821
10544
neutral or dissatisfied
satisfied
Predicted label
14000
12000
10000
8000
6000
4000
2000

# CONCLUSIONES

- Muy buen desempeño.
- Mejoras:
  - Minimizar los falsos negativos ajustando el umbral de decisión.
  - Optimizar los pesos en la función de pérdida.

# MUCHAS GRACIAS