# Machine Learning Pipeline Report

This report explains the process and implementation of a machine-learning pipeline to analyze an event dataset, define churn, engineer features, train a model, and interpret the results. Below is a detailed step-by-step breakdown:  
  
---  
  
 1. Importing Libraries  
The following libraries were used:  
  
  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix  
from sklearn.preprocessing import LabelEncoder  
from google.colab import drive  
  
  
- Pandas (pd): For handling data frames, including reading, manipulating, and analyzing data.  
- NumPy (np): For numerical operations and handling arrays.  
- Seaborn (sns) and Matplotlib (plt): For visualizing data through plots and graphs.  
- Scikit-learn modules:  
 - train\_test\_split: Splits data into training and testing sets.  
 - RandomForestClassifier: Builds a random forest model for classification.  
 - classification\_report and confusion\_matrix: Evaluates model performance.  
 - LabelEncoder: Encodes categorical variables into numeric values.  
- Google Colab Drive: To load datasets directly from Google Drive.  
  
---  
  
 2. Mounting Google Drive and Loading the Dataset  
The dataset is loaded from a specified Google Drive path:  
  
  
  
drive.mount('/content/drive')  
data\_path = '/content/drive/MyDrive/Colab Notebooks/events.csv'  
df = pd.read\_csv(data\_path)  
  
  
- The drive.mount method connects Google Drive to the Colab environment.  
- pd.read\_csv reads the dataset into a Pandas DataFrame.  
  
---  
  
 3. Data Cleaning  
The load\_and\_clean\_data function handles missing values, duplicates, and ensures correct data types:  
  
  
def load\_and\_clean\_data(filepath):  
 data = pd.read\_csv(filepath)  
 print("Missing Values before cleaning:\n", data.isnull().sum())  
 data = data.dropna()  
 print("Duplicate Rows before cleaning:", data.duplicated().sum())  
 data = data.drop\_duplicates()  
 data['event\_time'] = pd.to\_datetime(data['event\_time'])  
 print("Data Types after cleaning:\n", data.dtypes)  
 return data  
  
  
- Missing Values: Rows with missing values are removed using dropna().  
- Duplicates: Duplicate rows are identified with duplicated() and removed using drop\_duplicates().  
- Event Time Parsing: The event\_time column is converted to a datetime format for time-based analysis.  
  
---  
  
 4. Exploratory Data Analysis (EDA)  
The exploratory\_data\_analysis function provides insights into the dataset through visualizations:  
  
  
def exploratory\_data\_analysis(data):  
 sns.countplot(x='event\_type', data=data, order=data['event\_type'].value\_counts().index)  
 plt.title('Event Type Distribution')  
 plt.show()  
  
 data['date'] = data['event\_time'].dt.date  
 daily\_events = data.groupby(['date', 'event\_type']).size().unstack(fill\_value=0)  
 daily\_events.plot(kind='line', figsize=(12, 6))  
 plt.title('Daily Event Distribution')  
 plt.show()  
  
 sns.countplot(y='brand', data=data, order=data['brand'].value\_counts().head(10).index)  
 plt.title('Top 10 Popular Brands')  
 plt.show()  
  
 sns.countplot(y='category\_code', data=data, order=data['category\_code'].value\_counts().head(10).index)  
 plt.title('Top 10 Popular Categories')  
 plt.show()  
  
  
- Event Distribution: Visualized with countplot.  
- Daily Trends: Grouped by date and event type, and plotted as a line chart.  
- Brand and Category Popularity: The top 10 brands and categories are displayed using horizontal bar plots.  
  
---  
  
 5. Defining Churn  
The define\_churn function identifies users likely to churn:  
  
  
def define\_churn(data):  
 data['is\_purchase'] = data['event\_type'] == 'purchase'  
 churn\_threshold = pd.Timestamp(data['event\_time'].max()) - pd.Timedelta(days=30)  
 user\_last\_purchase = data[data['is\_purchase']].groupby('user\_id')['event\_time'].max()  
 churned\_users = user\_last\_purchase[user\_last\_purchase < churn\_threshold].index  
 data['is\_churn'] = data['user\_id'].isin(churned\_users)  
 print("Churned Users Count:", len(churned\_users))  
 return data  
  
  
- Definition: A user is considered churned if they have not made a purchase in the last 30 days.  
- Implementation:  
 - Identify purchases using is\_purchase.  
 - Calculate the last purchase for each user.  
 - Mark users as churned if their last purchase is older than 30 days.  
  
---  
  
 6. Feature Engineering  
The feature\_engineering function creates features for modeling:  
  
  
def feature\_engineering(data):  
 features = pd.DataFrame()  
 features['total\_events'] = data.groupby('user\_id').size()  
 features['total\_spend'] = data[data['event\_type'] == 'purchase'].groupby('user\_id')['price'].sum().fillna(0)  
 time\_diffs = data.groupby('user\_id')['event\_time'].diff().dt.total\_seconds()  
 features['avg\_time\_between\_events'] = time\_diffs.groupby(data['user\_id']).mean().fillna(0)  
 features['is\_churn'] = features.index.isin(data[data['is\_churn']]['user\_id'])  
 return features  
  
  
- Examples:  
 - Total events per user.  
 - Total spending by users who made purchases.  
 - Average time between events per user.  
 - Churn label for each user.  
  
---  
  
 7. Modeling  
The train\_model function builds and evaluates the machine-learning model:  
  
  
def train\_model(features):  
 X = features.drop(columns=['is\_churn'])  
 y = features['is\_churn']  
 le = LabelEncoder()  
 for col in X.select\_dtypes(include=['object']).columns:  
 X[col] = le.fit\_transform(X[col])  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
 model = RandomForestClassifier(random\_state=42)  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))  
 print("Classification Report:\n", classification\_report(y\_test, y\_pred))  
 return model  
  
  
- Data Preparation: Splits features and target into training and testing sets.  
- Model: Random Forest Classifier.  
- Evaluation: Confusion matrix and classification report are generated.  
  
---  
  
 8. Model Interpretation  
The model\_interpretation function identifies important features:  
  
  
def model\_interpretation(model, features):  
 feature\_importances = pd.Series(model.feature\_importances\_, index=features.drop(columns=['is\_churn']).columns)  
 feature\_importances.sort\_values(ascending=False).head(10).plot(kind='bar', figsize=(10, 6))  
 plt.title('Top 10 Feature Importances')  
 plt.show()  
  
  
- Feature Importances: Extracted from the Random Forest model and visualized.  
  
---  
  
 9. Recommendations  
The recommendations function suggests actionable insights:  
  
  
def recommendations():  
 print("1. Target users with high activity but no purchases using personalized offers.")  
 print("2. Identify users with long time gaps between sessions and send re-engagement campaigns.")  
 print("3. Analyze popular brands and categories to optimize inventory and marketing strategies.")  
  
  
- Suggestions:  
  
 - Re-engage users at risk of churning.  
 - Focus marketing efforts on popular brands and categories.  
 - Offer personalized deals to increase conversions.  
  
---  
  
 Conclusion  
This pipeline effectively identifies churned users, builds predictive models, and generates actionable insights to improve user retention and optimize business strategies.