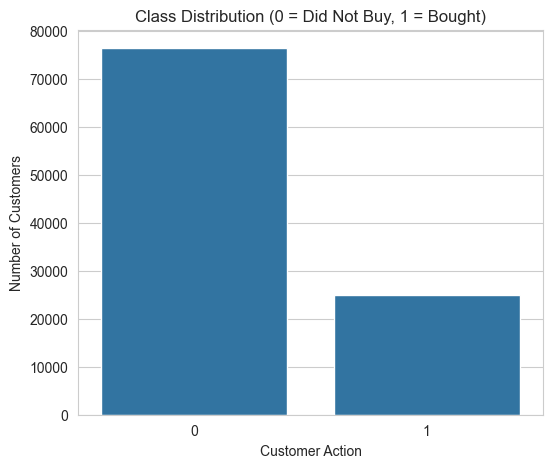
### **Project Report: Customer Purchase Prediction**

#### 

#### **Getting to know the Data**

The journey started with a dataset of customer information. This data had 22 features (F1 to F22) about different customers—things like their demographics and past actions. The goal was to predict the C column, which showed whether a customer bought a product (1) or not (0).

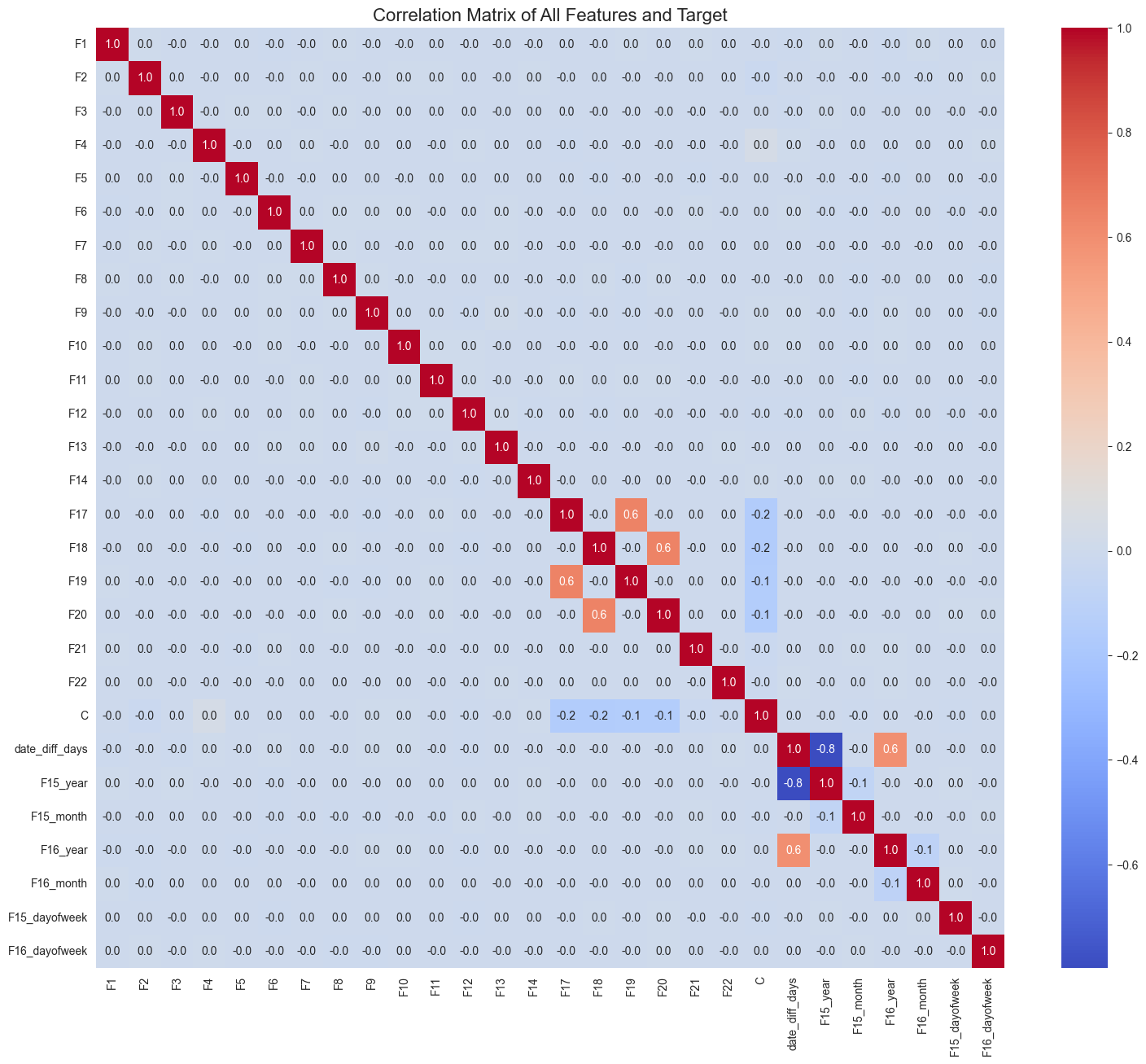
Looking at the data, one big challenge stood out immediately: it was **severely imbalanced**.



#### **Transforming the Raw Data into Clues**

Raw data is rarely useful on its own. It needs to be shaped into meaningful clues for the model.

* **Making Dates Make Sense:** Two features, F15 and F16, were dates. A model doesn't understand "July 4th, 2023." So, these dates were transformed into numbers the model could use, like the number of days between them (date\_diff\_days), the year, the month, and the day of the week. This helps the model spot patterns like "people tend to buy more on Fridays" or "sales dropped in 2022."
* **Removing "Echoes":** Sometimes, two features carry the same information, creating an echo that can confuse the model. A check was done for any features that were very highly correlated (a score > 0.7), and these were removed to keep the input clean and focused.

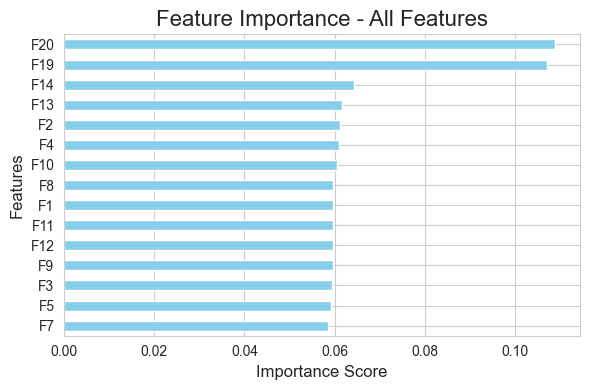


* **Leveling the Playing Field:** The features had very different scales. One feature might be a customer's age (e.g., 35), while another might be their income (e.g., 80,000). To prevent the model from thinking income is more important just because the number is bigger, a StandardScaler was used. This puts all features on the same scale.

#### **The Game Plan: A Hybrid Model**

A single model wasn't quite smart enough for this tricky data. So, a two-stage hybrid approach was used, making two models work together as a team.

1. **The Guide (Random Forest):** The first model was a **Random Forest**. Think of it as a team of experts. After they all analyze the data, they can be asked which clues (features) were the most helpful. The Random Forest was used to rank all the features, and then only the **top 15** were selected. This works like an intelligent filter, clearing out the noise so the main model can focus on what matters.



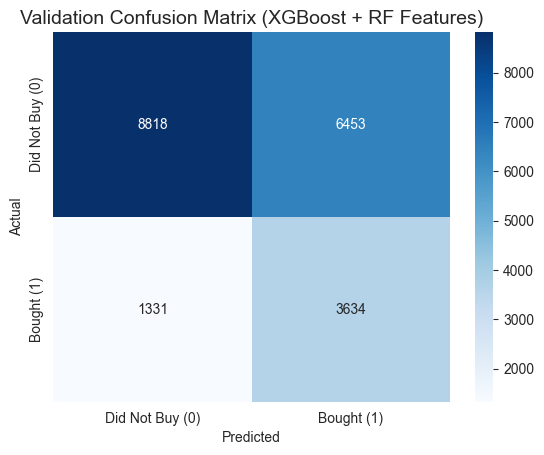
1. **The Expert (XGBoost):** The top 15 features were then handed over to the expert predictor: an **XGBoost** model. XGBoost is famous for its accuracy. To tackle the imbalanced data problem, it was given a special instruction: scale\_pos\_weight. This tells the model, "The 'buy' cases are rare, so treat them as extra important." This forces the model to work much harder to learn the patterns of the small group of buyers.

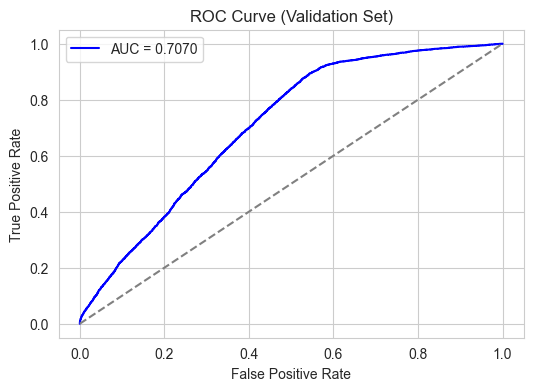
This team-based approach combines the filtering smarts of Random Forest with the predictive power of XGBoost.

#### **Measuring What Matters**

With imbalanced data, **accuracy** is a trap. A model could be 95% accurate but still be useless if it never finds a single buyer. So, better metrics were needed to judge success:

* **Recall:** This was the most important metric. It asks a simple question: *Of all the people who actually bought the product, how many did the model find?* A high recall is key to not missing out on sales.
* **Precision:** This asks: *When the model says someone will buy, how often is it right?*
* **Confusion Matrix:** This is a simple scorecard that visually breaks down the model's performance. It shows exactly how many predictions were right, how many were wrong, and what kinds of mistakes were made.





#### **The Final Result**

After all the experiments, the final hybrid model produced an excellent, balanced result.

* **Recall (for Buyers):** **73%**
* **Overall Accuracy:** **61%**
* **AUC-ROC: 70.7%**

It means the model can find **nearly 3 out of every 4 customers** who will actually make a purchase. This provides a high-quality, actionable list of leads for any marketing effort.

**Script**

import pandas as pd

import numpy as np

import xgboost as xgb

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

# Step 1: Load Data

train\_df = pd.read\_csv("Dataset.txt", sep="\t", index\_col="Index")

test\_df = pd.read\_csv("Dataset\_test.txt", sep="\t", index\_col="Index")

# Step 2: Date Features

def create\_date\_features(df):

df["F15"] = pd.to\_datetime(df["F15"])

df["F16"] = pd.to\_datetime(df["F16"])

df["date\_diff\_days"] = (df["F16"] - df["F15"]).dt.days

df["F15\_year"] = df["F15"].dt.year

df["F15\_month"] = df["F15"].dt.month

df["F15\_dayofweek"] = df["F15"].dt.dayofweek

df["F16\_year"] = df["F16"].dt.year

df["F16\_month"] = df["F16"].dt.month

df["F16\_dayofweek"] = df["F16"].dt.dayofweek

df.drop(["F15", "F16"], axis=1, inplace=True)

return df

train\_df = create\_date\_features(train\_df)

test\_df = create\_date\_features(test\_df)

# Step 3: Remove highly correlated features

corr\_matrix = train\_df.drop(columns=["C"]).corr().abs()

upper\_triangle = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

high\_corr\_features = [col for col in upper\_triangle.columns if any(upper\_triangle[col] > 0.7)]

train\_df.drop(columns=high\_corr\_features, inplace=True)

test\_df.drop(columns=high\_corr\_features, inplace=True)

# Step 4: RF feature selection

X = train\_df.drop("C", axis=1)

y = train\_df["C"]

rf = RandomForestClassifier(random\_state=42)

rf.fit(X, y)

rf\_feature\_importances = pd.Series(rf.feature\_importances\_, index=X.columns)

top\_features = rf\_feature\_importances.sort\_values(ascending=False).head(15).index.tolist()

X = X[top\_features]

X\_test = test\_df[top\_features]

# Step 5: Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_test\_scaled = scaler.transform(X\_test)

# Step 6: Final XGBoost training

ratio = y.value\_counts()[0] / y.value\_counts()[1]

model = xgb.XGBClassifier(

objective="binary:logistic",

eval\_metric="logloss",

scale\_pos\_weight=ratio,

random\_state=42

)

model.fit(X\_scaled, y)

# Step 7: Predictions

pd.DataFrame({"Class": model.predict(X\_scaled)}, index=X.index).to\_csv("training\_predictions.txt", sep="\t")

pd.DataFrame({"Class": model.predict(X\_test\_scaled)}, index=X\_test.index).to\_csv("test\_predictions.txt", sep="\t")