

✓ Project Description:

This project demonstrates the complete end-to-end machine learning pipeline for predicting customer churn in a telecom domain using a combination of Apache Spark (via Databricks) and Scikit-learn. It highlights data engineering, data science, and MLOps best practices, leveraging both big data tools and local interpretability libraries.

The solution is designed for scalable processing on large datasets, model training using Spark MLlib, and in-depth evaluation and visualization using Scikit-learn and Pandas

✓ Key Components:

1. Data Ingestion & Exploration

- Loaded Delta Lake table `mlops.mlops_schema.telco_customer_churn` using Spark.
- Performed exploratory data analysis (EDA) with SQL and Pandas API on Spark.
- Visualized service usage distributions (e.g., `InternetService`) via pie charts.

2. Data Cleaning & Feature Engineering

- Handled missing values, type casting, and trimmed textual inconsistencies.
- Created new features like `num_optional_services` to enhance signal strength.
- Standardized columns like `SeniorCitizen` from binary to categorical ("Yes"/"No").

3. Data Splitting for ML Lifecycle




- Split the dataset into train, validate, and test sets using random sampling with seeds for reproducibility.
- Persisted the feature-engineered dataset to a managed Delta table for reuse.

4. Modeling with Spark MLlib

- Created an ML pipeline with `StringIndexer`, `VectorAssembler`, and `LogisticRegression`.
- Trained the model on distributed Spark clusters for scalability.
- Evaluated using ROC AUC in Spark environment.

5. Scikit-learn Modeling & Visual Analytics

- Converted Spark `DataFrame` to Pandas for deeper ML insights.
- Performed one-hot encoding on categorical features.
- Trained a local logistic regression model for interpretability.
- Visualized:

-  Feature importance (bar chart)
-  ROC curve
-  Confusion matrix

✓ Outcomes:

- Achieved high AUC scores indicating strong model performance.
- Identified key churn drivers such as MonthlyCharges, Contract, and Optional Services.
- Built a reusable and explainable MLOps-ready pipeline combining Spark scale and Scikit-learn flexibility.

✓ Python Libraries & Packages

Package	Purpose
pyspark	Distributed data processing, ML pipelines (MLlib), Spark SQL queries
pyspark.sql.functions	Feature engineering, column manipulation, random splitting, filtering
mlflow	Experiment tracking, model autologging (optional for MLOps)
pandas	Local data manipulation and preprocessing for Scikit-learn
matplotlib	Data visualization (ROC curve, confusion matrix, etc.)
seaborn	Enhanced visualizations (e.g., feature importance barplots)
scikit-learn	Machine learning model training, evaluation, feature engineering
Planform	DATABRICKS with MLOps installed

✓ Steps to train the algorithm

Step-1: Installing mlflow

```
%pip install mlflow==2.22.0
```

```
%restart_python
```

```
import mlflow print(mlflow.__version__)
```

STEP-2: Reading Customer Churn table into data frame

```
telcoDF = spark.read.table("mlops.mlops_schema.telco_customer_churn")
display(telcoDF)
```

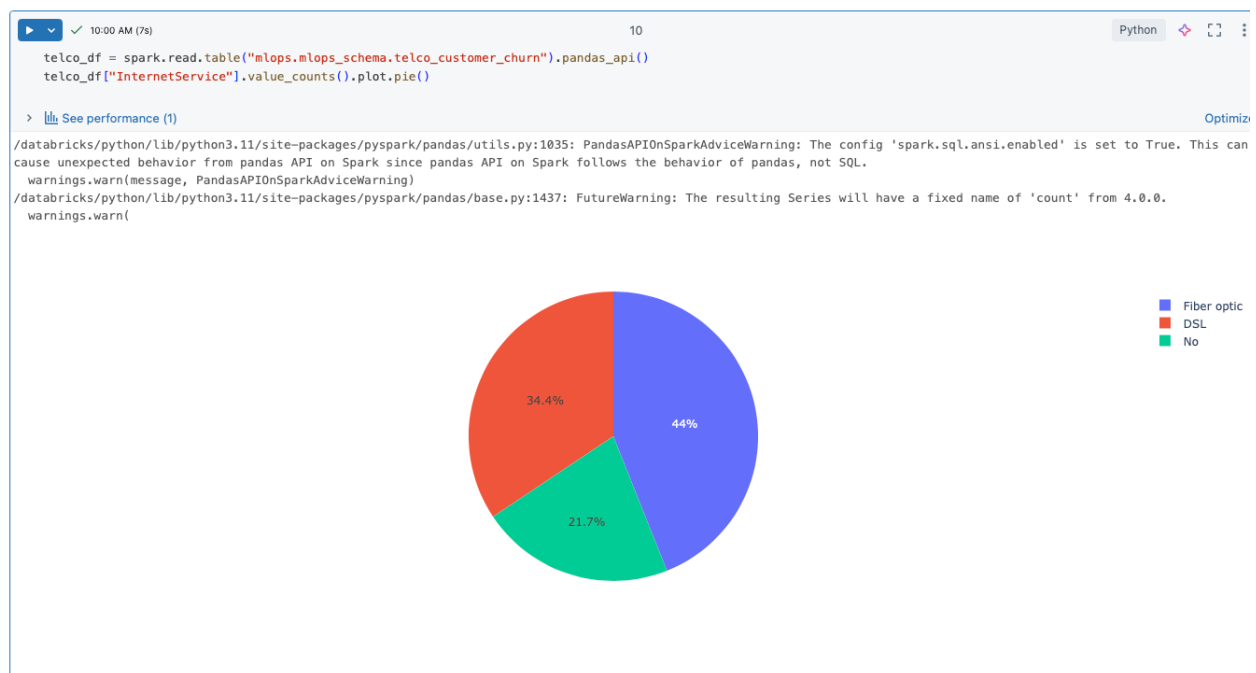
STEP-3: Analyze the data and prepare features

```
SELECT * FROM mlops.mlops_schema.telco_customer_churn
```

Here just look at the table and understand if there anything missing or want to add

To understand the different types of contracts, Plotting the pie chart based on the count for each of the service types under InternetService column.

```
telco_df =
spark.read.table("mlops.mlops_schema.telco_customer_churn").pandas_api()
telco_df["InternetService"].value_counts().plot.pie()
```



STEP-4: Prepare the dataset which will be trained later

Few keys things to understand from the below code

- Converted the dataframe to pandas as I have been using databricks free tier and it restricts many features and pandas helps us better.
- Formatting the table to make it to be trained against the model

```
import pyspark.sql.functions as F
from pyspark.sql import DataFrame

def clean_churn_features(dataDF: DataFrame) -> DataFrame:

    # converty to pandas on spark dataframe
    data_psdof = dataDF.pandas_api()

    #convert some columns

    data_psdof["SeniorCitizen"] = data_psdof["SeniorCitizen"].astype("string")
    data_psdof["SeniorCitizen"] = data_psdof["SeniorCitizen"].map({"1": "Yes",
"0" : "No"})

    # check for leading and trailing spaces and convert it to float if not
null. if null then 0

    data_psdof["TotalCharges"] = data_psdof["TotalCharges"].apply(lambda x:
float(x) if x.strip() else 0)

    #Fill some missing numerical valies with o

    data_psdof = data_psdof.fillna({"tenure": 0.0})
    data_psdof = data_psdof.fillna({"MonthlyCharges": 0.0})
    data_psdof = data_psdof.fillna({"TotalCharges": 0.0})

    # Count the number of optional services enabled for each row of customers
    def sum_optional_services(df):

        cols = ["OnlineSecurity", "OnlineBackup", "DeviceProtection",
"TechSupport",
                "StreamingTV", "StreamingMovies"]

        return sum(map(lambda c: (df[c] == "Yes"), cols))
        return sum(map(lambda c: (df[c] == "Yes"), cols))
```

```
data_psdF["num_optional_services"] = sum_optional_services(data_psdF)

#Return the cleaned Spark dataframe
return data_psdF.to_spark()
```

```
churn_features = clean_churn_features(telcoDF)
```

STEP5: Split or categorize the data for train, validate & test the model.

Just remember that every dataset needs to be split either as 80/20% or 70/30 to train the model and to test the result.

```
train_ratio, val_ratio, test_ratio = 0.7, 0.2, 0.1

churn_features = (churn_features.withColumn("random", F.rand(seed=42))
                  .withColumn("split",
                              F.when(F.col("random") <
train_ratio, "train")
                              .when(F.col("random") <
train_ratio + val_ratio, "validate")
                              .otherwise("test")))
                  .drop("random"))
```

STEP-6: Write the final cleaned dataframe to a table for later user.

```
(churn_features.write.mode("overwrite")
 .option("overwriteSchema", "true")
 .saveAsTable("mlops.mlops_schema.churn_features"))
```

STEP-7: The actual show starts from here. Train the dataset against the model.

I have used 'LOGISTIC REGRESSION' model to find the churn rate for customers, because it is a Binary Classification Problem

- The target variable churn has two classes: **Yes** or **No**.
- **Logistic Regression** is **specifically designed** to handle binary classification tasks by estimating the probability of class membership.

Example:

It outputs a probability between 0 and 1 that a customer will churn, which can acts as threshold(e.g., > 0.5) to classify as churn or not.

```
import pandas as pd
import mlflow

# Automatically logs model training information to MLflow without manually specifying
every detail.
mlflow.autolog()

# Load Spark table
spark_df = spark.read.table("mlops.mlops_schema.telco_customer_churn")

# Clean with your function
cleaned_df = clean_churn_features(spark_df)

# Convert to pandas (on driver)
pdf = cleaned_df.toPandas()
print(pdf.columns.tolist())

pdf.columns = pdf.columns.str.strip().str.lower()
print(pdf.columns.tolist())

# Encode churn label→ create a label column by just giving 1 to yes and 0 to
no. To create a binary label column (label) from the churn column in your
Pandas DataFrame pdf. This is essential for supervised machine learning,
where the target variable (label) must be numeric (e.g., 0 or 1).

pdf["label"] = pdf["churn"].str.strip().str.lower().map(lambda x: 1 if x ==
"yes" else 0)
```

```
# One-hot encode categorical variables. These three columns are used by model
and they need to be converted to binary values as 0 or 1. The below code
converts them into binary values.
```

```
cat_cols = ["gender", "contract", "internetservice"] # update as needed
pdf = pd.get_dummies(pdf, columns=cat_cols, drop_first=True)
```

```
# Select features columns which are essential for model training
features = ["tenure", "monthlycharges", "totalcharges",
"num_optional_services"] + \
    [col for col in pdf.columns if col.startswith(tuple(cat_cols))]
```

```
#Defining X & Y variables. X is list of feature columns used by model and Y
is the label/churn information.
```

```
X = pdf[features]
y = pdf["label"]
```

```
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=42)
```

```
# Train model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
```

```
#Calling the Logistic Regression model for training
```

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
# It gets the predicted probabilities of the test data belonging to class 1
(i.e., churn = 1). You don't just want the model's "yes or no" decision – you
want to know how confident it is, which is crucial for metrics like ROC AUC.
```

```
y_pred_proba = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"ROC AUC: {roc_auc:.4f}")
```

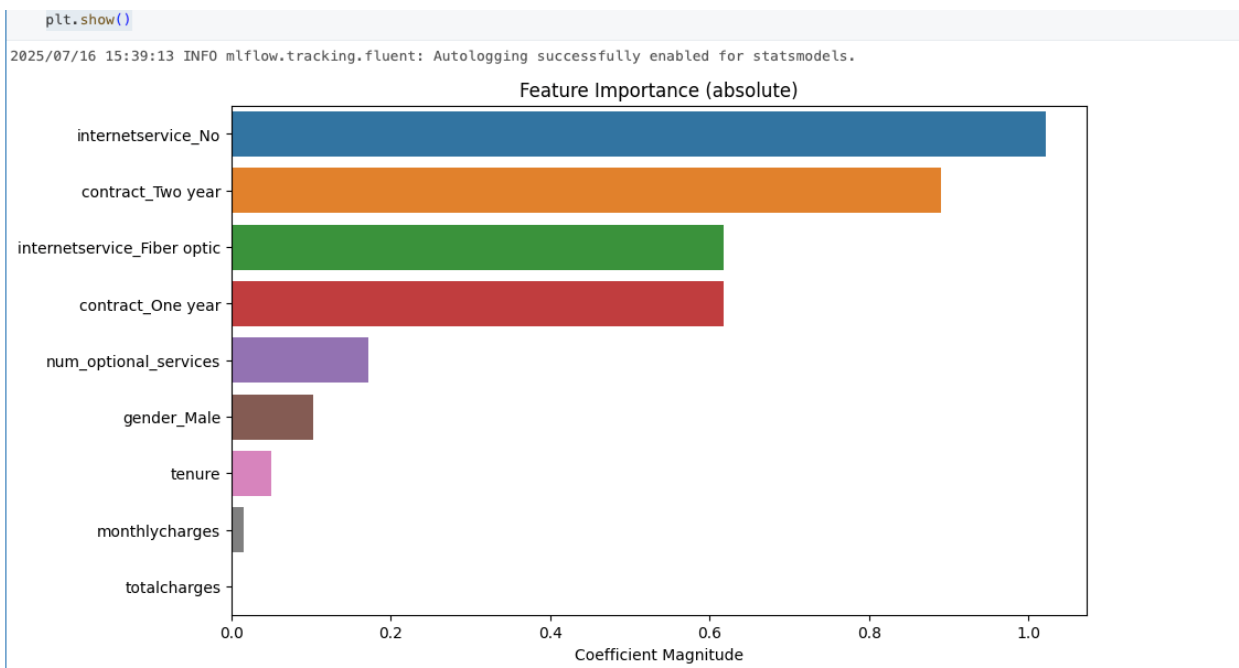
```

import matplotlib.pyplot as plt
import seaborn as sns

importance = pd.Series(model.coef_[0], index=X.columns)
plt.figure(figsize=(10, 6))
sns.barplot(x=importance.abs().sort_values(ascending=False),
y=importance.abs().sort_values(ascending=False).index)
plt.title("Feature Importance (absolute)")
plt.xlabel("Coefficient Magnitude")
plt.show()

```

This code tells you which features has strong influence on the churn rate. Either positive or negative, it gives which types of features has more impact on the churn decision.



```

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Compute feature importances (absolute coefficient values)
importance = pd.Series(model.coef_[0], index=X.columns).sort_values(key=abs,
ascending=False)

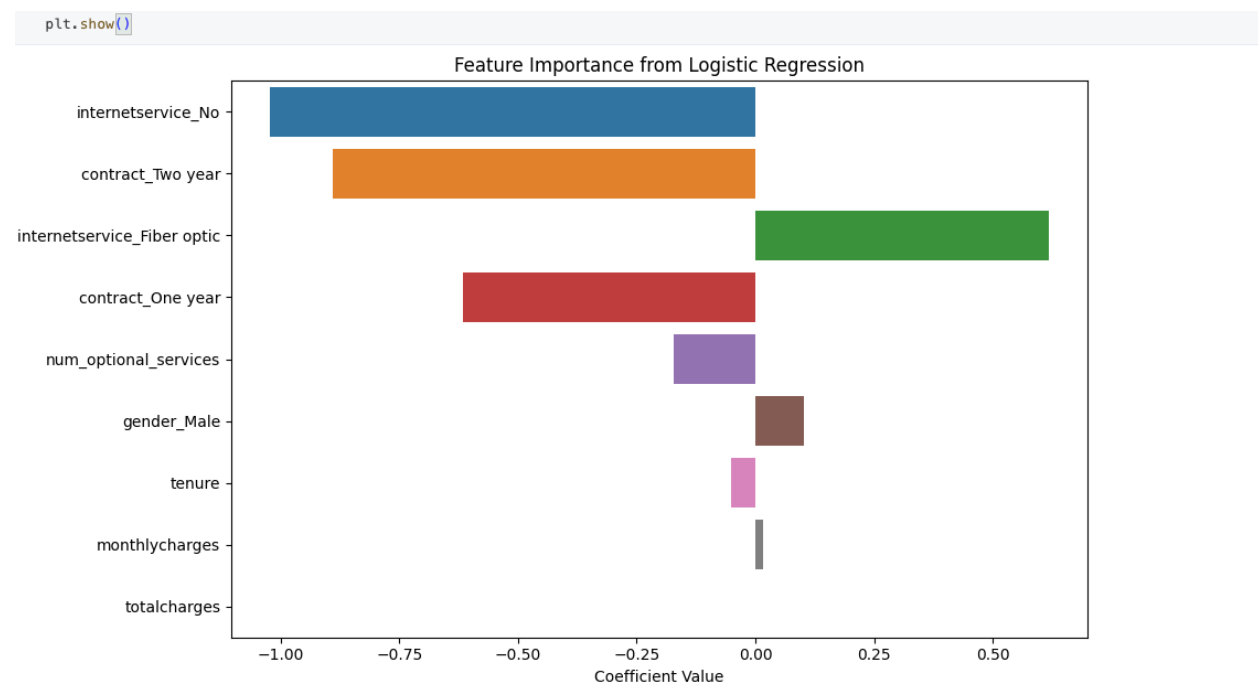
# Plot

```



```
plt.figure(figsize=(10, 6))
sns.barplot(x=importance.values, y=importance.index)
plt.title("Feature Importance from Logistic Regression")
plt.xlabel("Coefficient Value")
plt.tight_layout()
plt.show()
```

This visual show which of features will either positive or negative impact and how much. Looking at below, the internet service_no customers are less likely to churn as the values are negative which are higher. Similarly, the customer with fiber optic are more likely to churn because the coefficient value is higher positive.



```
from sklearn.metrics import roc_curve, auc

y_pred_proba = model.predict_proba(X_test)[: , 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color="blue", lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
```

```
plt.plot([0, 1], [0, 1], color="gray", lw=2, linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Churn Prediction")
plt.legend(loc="lower right")
plt.grid()
plt.tight_layout()
plt.show()
```

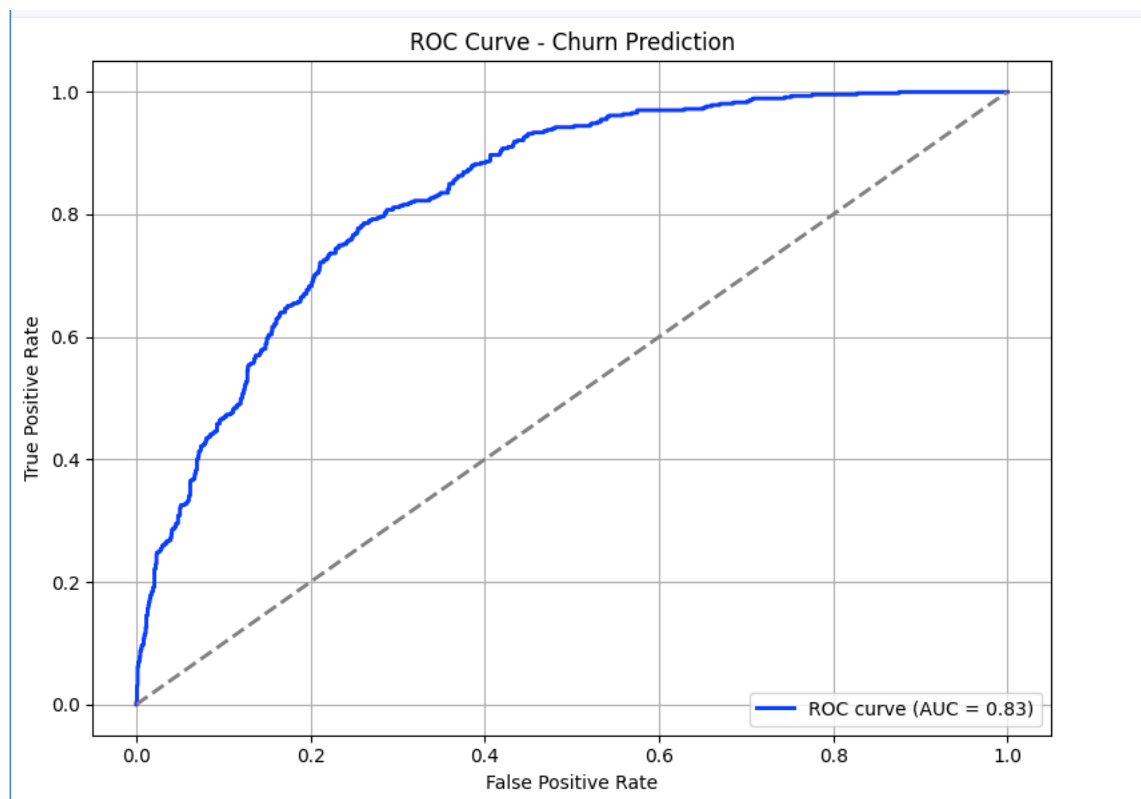
ROC = Receiver Operating Characteristic.

AUC = Area Under the ROC Curve.

Uses the trained model to get **predicted probabilities** for the **positive class** (churn = 1).

The result is an array of values between 0 and 1

1. You evaluated your logistic regression model using the ROC curve.
2. AUC = **0.83** shows your model is **performing well**.
3. ROC curve is a great way to visualize **the model's ability to separate classes** regardless of threshold



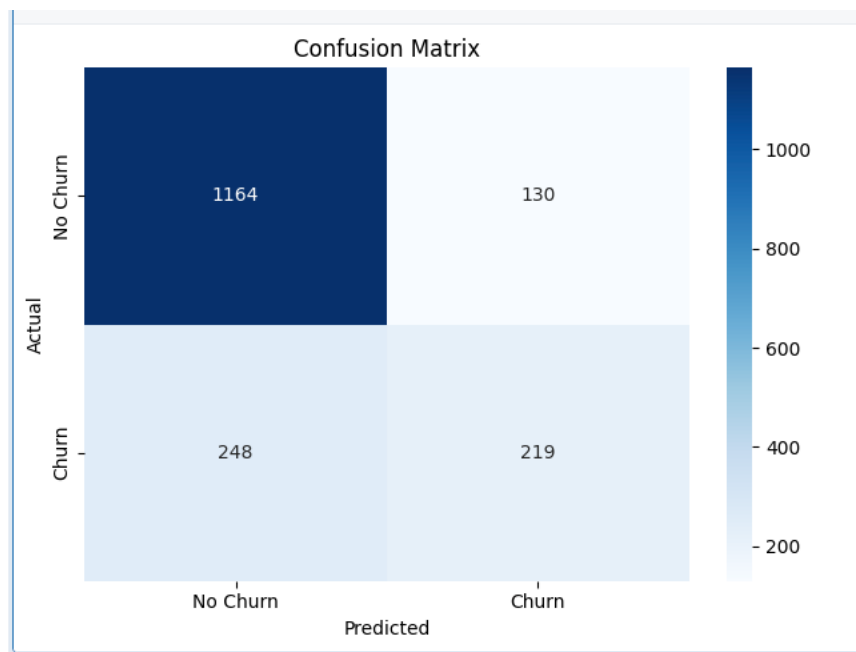
```

from sklearn.metrics import confusion_matrix

y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["No Churn",
"Churn"], yticklabels=["No Churn", "Churn"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

```



Our Confusion Matrix Output:

	Predicted: No Churn	Predicted: Churn
Actual: No Churn	1164 (TN)	130 (FP)
Actual: Churn	248 (FN)	219 (TP)

Let's label them:

- **TN (True Negative)** = 1164: Correctly predicted non-churners ✓
- **FP (False Positive)** = 130: Predicted churn, but didn't churn ✗
- **FN (False Negative)** = 248: Predicted no churn, but actually churned ✗
- **TP (True Positive)** = 219: Correctly predicted churners ✓

✓ Summary:

- You correctly predicted:
 - **219 churners**
 - **1164 non-churners**
- But missed:
 - **248 churners**
 - **130 non-churners** were falsely flagged

Final Step: Business Insights on Model Prediction:

Overall, this shows **high accuracy but moderate recall** — which might need adjustment if **catching churners is the top priority**.