Final Report: Customer Churn Prediction for a Telecom Company

By: Samarth Vedi

Project Definition

The main goal of this project is to use historical customer data to predict when a telecom company's customers will leave. Targeted retention tactics can help the company keep customers that are likely to leave by figuring out which ones. This project has strategic parts like gathering data, preprocessing it, feature engineering, building models, evaluating them, and deploying them.

Novelty and Importance

Importance

The project is important because it solves a major business problem: retaining customers. It's cheaper to keep people you already have than to get new ones. The business can take steps to keep customers by figuring out how many customers are likely to leave. And this has a direct effect on both strategic planning and income optimization.

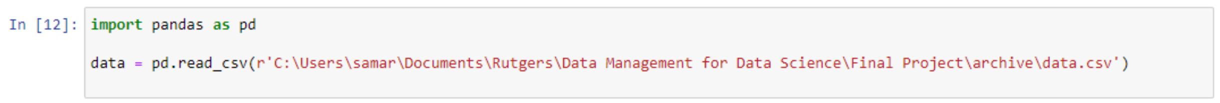
Existing Issues in Data Management Practices

1. Data Volume: Telecommunications companies create a lot of data, which makes it hard to store and process.
2. Data Quality: Predictive models can't work as well if the data they use isn't complete or consistent.
3. Privacy and Security: Making sure that info is kept private and safe while following the regulations.
4. Integration: Putting together data from different sources, like contact logs, billing systems, and CRM systems.

Progress and Contribution

1. Data Collection

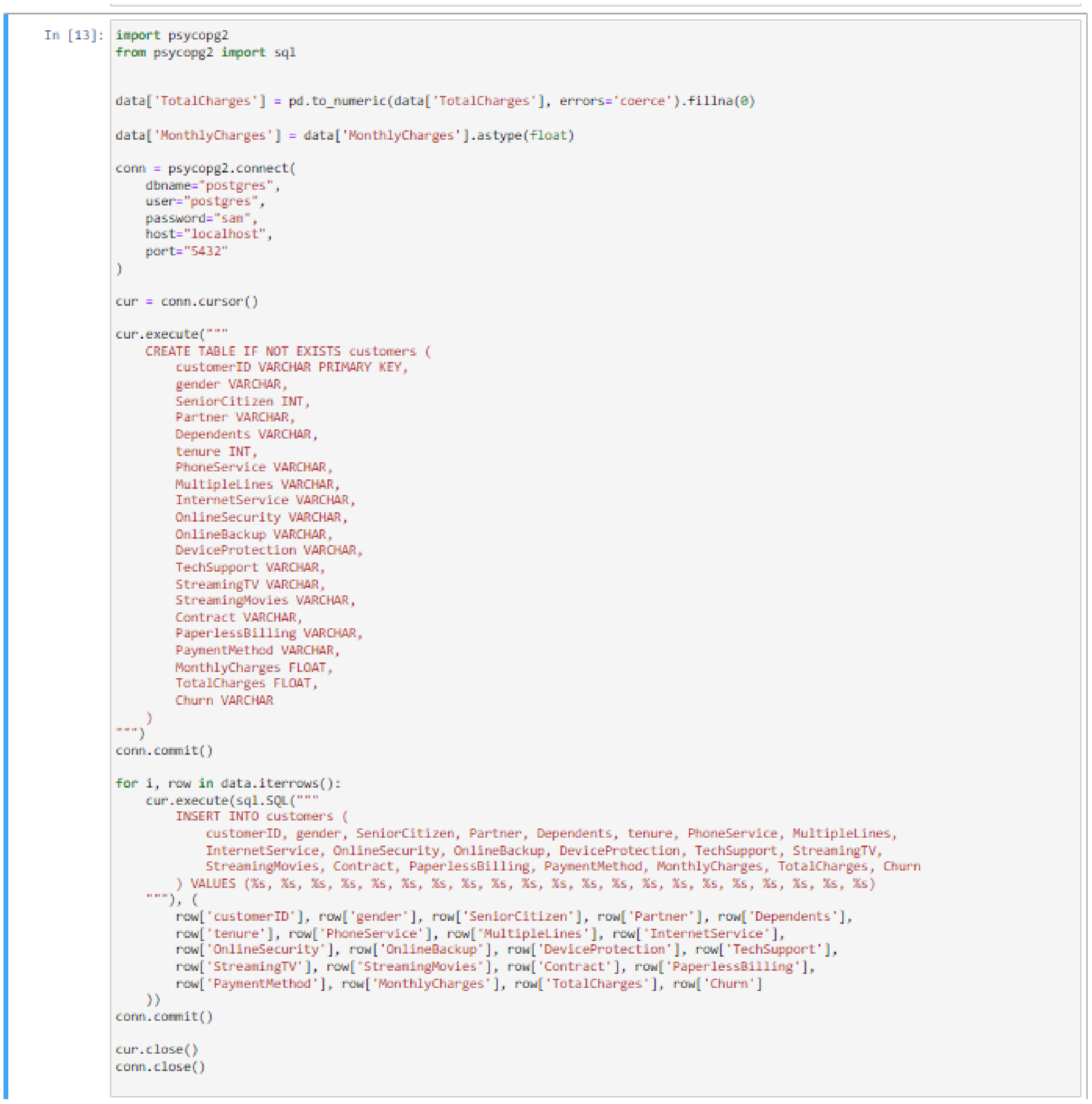
The dataset was collected from a CSV file containing various customer attributes such as tenure, monthly charges, total charges, and other relevant features.



```
In [12]: import pandas as pd

         data = pd.read_csv(r'C:\Users\samar\Documents\Rutgers\Data Management for Data Science\Final Project\archive\data.csv')
```

2. Data Storage

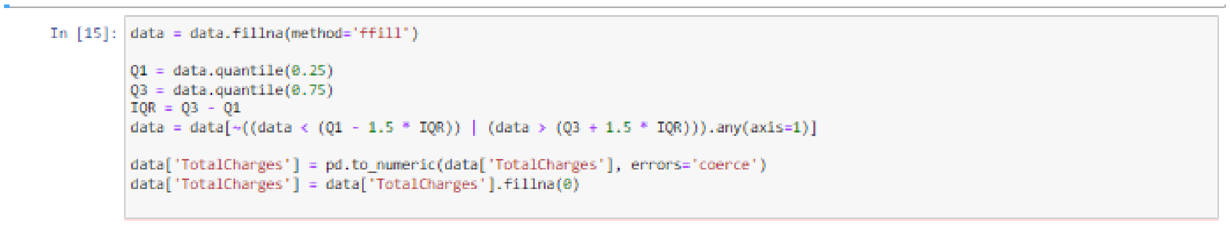
Data was stored in a PostgreSQL database for efficient retrieval and analysis.



```
In [13]: import psycopg2
         from psycopg2 import sql


         data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce').fillna(0)

         data['MonthlyCharges'] = data['MonthlyCharges'].astype(float)

         conn = psycopg2.connect(
             dbname="postgres",
             user="postgres",
             password="sam",
             host="localhost",
             port="5432"
         )

         cur = conn.cursor()

         cur.execute("""
             CREATE TABLE IF NOT EXISTS customers (
                 customerID VARCHAR PRIMARY KEY,
                 gender VARCHAR,
                 SeniorCitizen INT,
                 Partner VARCHAR,
                 Dependents VARCHAR,
                 tenure INT,
                 PhoneService VARCHAR,
                 MultipleLines VARCHAR,
                 InternetService VARCHAR,
                 OnlineSecurity VARCHAR,
                 OnlineBackup VARCHAR,
                 DeviceProtection VARCHAR,
                 TechSupport VARCHAR,
                 StreamingTV VARCHAR,
                 StreamingMovies VARCHAR,
                 Contract VARCHAR,
                 PaperlessBilling VARCHAR,
                 PaymentMethod VARCHAR,
                 MonthlyCharges FLOAT,
                 TotalCharges FLOAT,
                 Churn VARCHAR
             )
         """)
         conn.commit()

         for i, row in data.iterrows():
             cur.execute(sql.SQL("""
                 INSERT INTO customers (
                     customerID, gender, SeniorCitizen, Partner, Dependents, tenure, PhoneService, MultipleLines,
                     InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV,
                     StreamingMovies, Contract, PaperlessBilling, PaymentMethod, MonthlyCharges, TotalCharges, Churn
                 ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
             """), (
                 row['customerID'], row['gender'], row['SeniorCitizen'], row['Partner'], row['Dependents'],
                 row['tenure'], row['PhoneService'], row['MultipleLines'], row['InternetService'],
                 row['OnlineSecurity'], row['OnlineBackup'], row['DeviceProtection'], row['TechSupport'],
                 row['StreamingTV'], row['StreamingMovies'], row['Contract'], row['PaperlessBilling'],
                 row['PaymentMethod'], row['MonthlyCharges'], row['TotalCharges'], row['Churn']
             ))
         conn.commit()

         cur.close()
         conn.close()
```

3. Data Cleaning

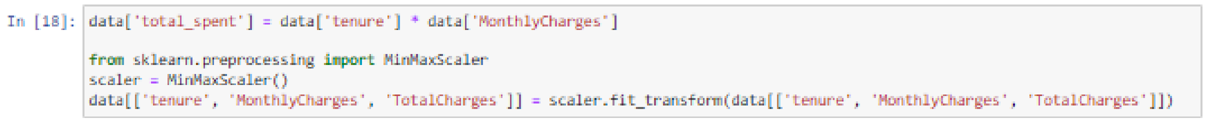
Data cleaning involved handling missing values and outliers.



```
In [15]: data = data.fillna(method='ffill')

         Q1 = data.quantile(0.25)
         Q3 = data.quantile(0.75)
         IQR = Q3 - Q1
         data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]

         data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
         data['TotalCharges'] = data['TotalCharges'].fillna(0)
```

4. Data Transformation

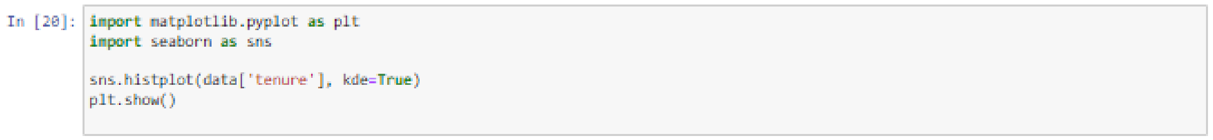
Feature engineering and normalization were performed to prepare the data for modeling.



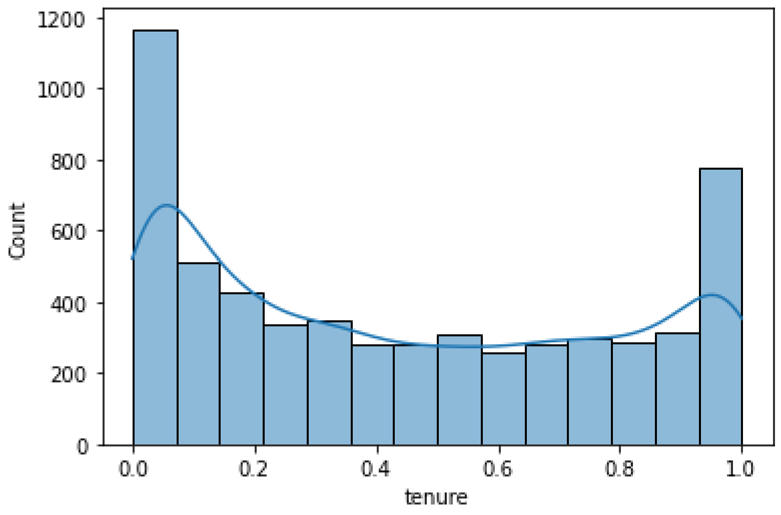
```
In [18]: data['total_spent'] = data['tenure'] * data['MonthlyCharges']

         from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler()
         data[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit_transform(data[['tenure', 'MonthlyCharges', 'TotalCharges']])
```

5. Exploratory Data Analysis (EDA)

EDA provided insights into the data distribution and relationships between features.

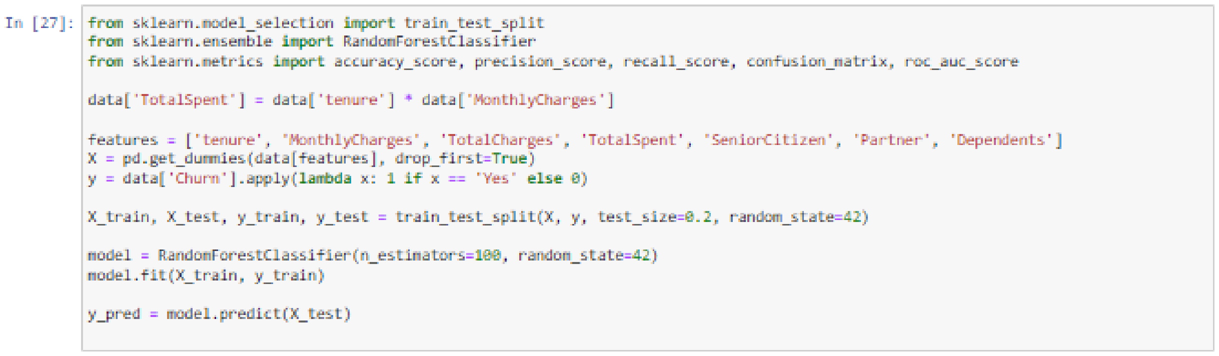


```
In [20]: import matplotlib.pyplot as plt
         import seaborn as sns

         sns.histplot(data['tenure'], kde=True)
         plt.show()
```



6. Model Building

A RandomForestClassifier was used to build the predictive model.



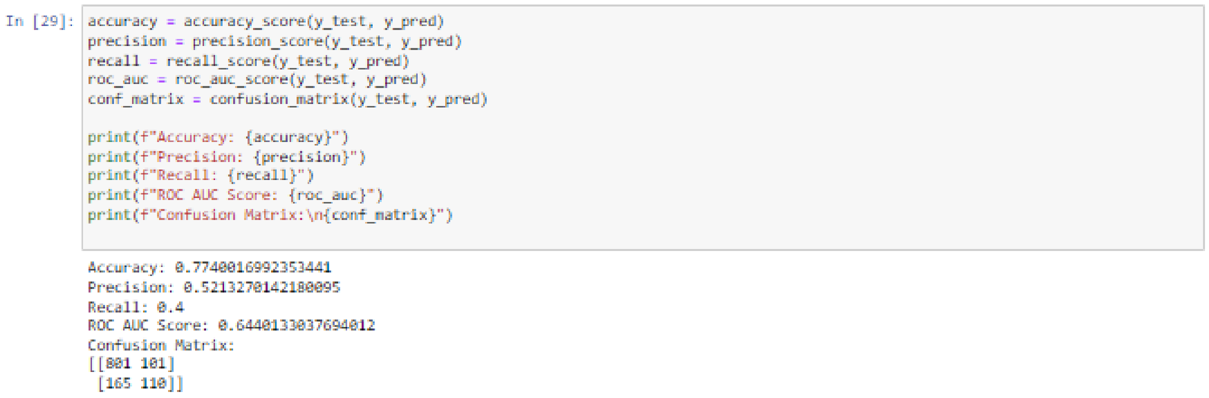
```
In [27]: from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, roc_auc_score

         data['TotalSpent'] = data['tenure'] * data['MonthlyCharges']

         features = ['tenure', 'MonthlyCharges', 'TotalCharges', 'TotalSpent', 'SeniorCitizen', 'Partner', 'Dependents']
         X = pd.get_dummies(data[features], drop_first=True)
         y = data['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

         model = RandomForestClassifier(n_estimators=100, random_state=42)
         model.fit(X_train, y_train)

         y_pred = model.predict(X_test)
```

7. Evaluation and Validation

The model was evaluated using various metrics.

Results:

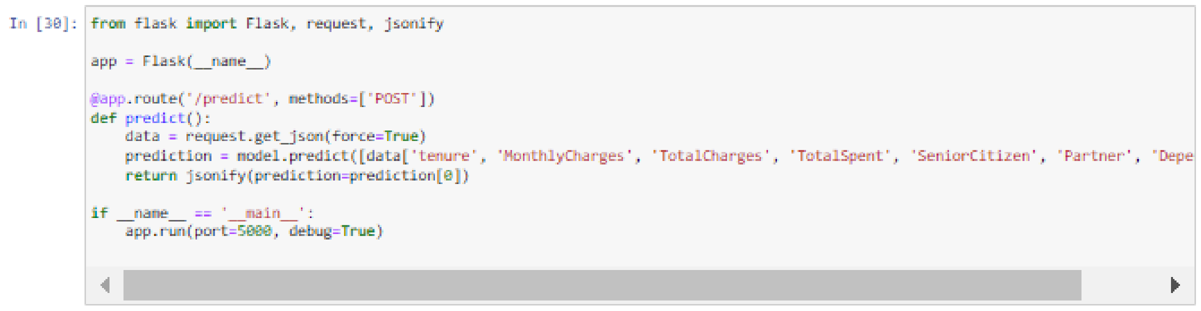
- **Accuracy**: 0.774
- **Precision**: 0.523
- **Recall**: 0.4
- **ROC AUC Score**: 0.644
- **Confusion Matrix**: [[881, 101], [165, 110]]



```
In [29]: accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred)
         recall = recall_score(y_test, y_pred)
         roc_auc = roc_auc_score(y_test, y_pred)
         conf_matrix = confusion_matrix(y_test, y_pred)

         print(f"Accuracy: {accuracy}")
         print(f"Precision: {precision}")
         print(f"Recall: {recall}")
         print(f"ROC AUC Score: {roc_auc}")
         print(f"Confusion Matrix:\n{conf_matrix}")

         Accuracy: 0.7740016992353441
         Precision: 0.5213270142180095
         Recall: 0.4
         ROC AUC Score: 0.6440133037694012
         Confusion Matrix:
         [[881 101]
          [165 110]]
```

8. Deployment

The model was deployed using Flask to create an API for real-time predictions.



```
In [30]: from flask import Flask, request, jsonify

         app = Flask(__name__)

         @app.route('/predict', methods=['POST'])
         def predict():
             data = request.get_json(force=True)
             prediction = model.predict([data['tenure', 'MonthlyCharges', 'TotalCharges', 'TotalSpent', 'SeniorCitizen', 'Partner', 'Depe
             return jsonify(prediction=prediction[0])

         if __name__ == '__main__':
             app.run(port=5000, debug=True)
```

Changes After Proposal

This plan was changed during the project in the following ways:

1. To make the TotalSpent feature, more feature engineering work had to be done.
2. The RandomForestClassifier's hyperparameters were changed to make the model work better.
3. The plan for deployment was improved to make sure that the Flask API could handle predictions in real time well.

Conclusion

A model was made for the project that had 77.4% accuracy that buyers will leave. The model could use some work to increase the precision and recall metrics. But, it's still a good place to start to find customers who are at risk of leaving. The model can be fine-tuned, other ways can be looked into, and more features can be added to make it work better in the future.