# Telecom Churn Prediction

**Step 1: Data Collection**

- **Data Source** – Kaggle

**Step 2: EDA and Data Cleaning**

- Conducted univariate analysis of numeric and categorical features.
- Performed bivariate analysis of numeric and categorical features with respect to the target column “Churn” using histograms and count plots (with hue="Churn").
- Identified outliers using boxplots and the IQR method.
- Analyzed skewness and kurtosis of numerical distributions.
- Explored correlations among independent numeric features.
- Performed bivariate analysis of numeric features using the groupby method to derive insights.
- Evaluated the correlation of numeric features with the categorical target column “Churn” using the pointbiserialr method from scipy.stats, to understand how numerical features relate to the target variable.
- Analyzed categorical feature distributions by “Churn” using the value_counts method.
- Conducted a detailed analysis of categorical features with respect to the target “Churn” using a combination of groupby and value_counts().
- Identified and removed less important features based on their impact on the model using the feature_importances_ attribute from a trained Random Forest classifier.

**Step 3: Feature Engineering**

- Combined similar categories where appropriate (e.g., merged “No internet service” into the “No” category).
- Identified ordinal and nominal features for tailored preprocessing.
- Created separate pipelines:
    - **Numeric Pipeline** – For numeric features: applied imputation and scaling.
    - **Ordinal Pipeline** – For ordinal features: determined feature hierarchy, applied imputation, and used an Ordinal Encoder.
    - **Nominal Pipeline** – For nominal features: applied imputation and One-Hot Encoding.

- Combined all pipelines using a ColumnTransformer.
- Split the dataset into training and testing sets.
- Applied the preprocessing and feature engineering pipeline to the data.

**Step 4: Machine Learning Model Training and Evaluation**

- Tested multiple classification algorithms and evaluated them based on precision, recall, and F1-score.
- Selected the Gradient Boosting algorithm for its balanced performance.
- In a telecom churn prediction context, recall was prioritized to minimize false negatives, which is crucial for identifying potential churners.

**Step 5: Hyperparameter Tuning of Gradient Boosting Algorithm**

- Performed hyperparameter tuning using GridSearchCV to explore various combinations of parameters.
- Applied cross-validation during tuning to assess generalization performance.
- Retrieved the best parameters from the grid search and trained the final model using these optimized values.

**Step 6: Improving Accuracy**

- **PCA (Principal Component Analysis):**
    - Experimented with different numbers of principal components.
    - Selected 10 components that captured over 90% of the total variance in the dataset.
    - Retrained the model using the PCA-transformed data.
- **Feature Importance:**
    - Assessed the contribution of each feature using the model's feature_importances_ attribute.
    - Created a DataFrame to rank features by importance.
    - Sorted features in descending order and identified the least impactful ones.
- **SHAP Analysis:**
    - Used SHAP (SHapley Additive exPlanations) and TreeExplainer to identify which feature values most influenced predictions.

- Extracted and visualized the features having the greatest impact on predictions.
- Created a DataFrame of features with their SHAP values, sorted by impact.
- Focused on features contributing only 1–2% to the prediction score.

**Step 6 (Continued): Model Refinement**

- Dropped low-importance features (noise) and retrained the model.
- Re-evaluated precision, recall, and F1-score after feature removal.
- If performance remained stable, finalized the reduced feature set for future modeling.

**Step 7: Model and Pipeline Serialization**

- Saved the trained model using Pickle.
- Also serialized the preprocessing pipeline using Pickle for reuse during inference.

**Step 8: Custom Predictions**

- Loaded the pickled model and preprocessor pipeline.
- Collected input data from users.
- Converted the input into a dictionary (key-value pairs).
- Transformed the dictionary into a DataFrame.
- Applied the preprocessing pipeline to the user input.
- Passed the processed data to the model for prediction.

**Step 9: End-to-End Application Development**

- Built an interactive web application using Streamlit.
- Integrated the model and preprocessor to enable real-time predictions.

**Step 10: Deployment**

- Deployed the Streamlit application using GitHub for version control and hosting.