Churn Analytics

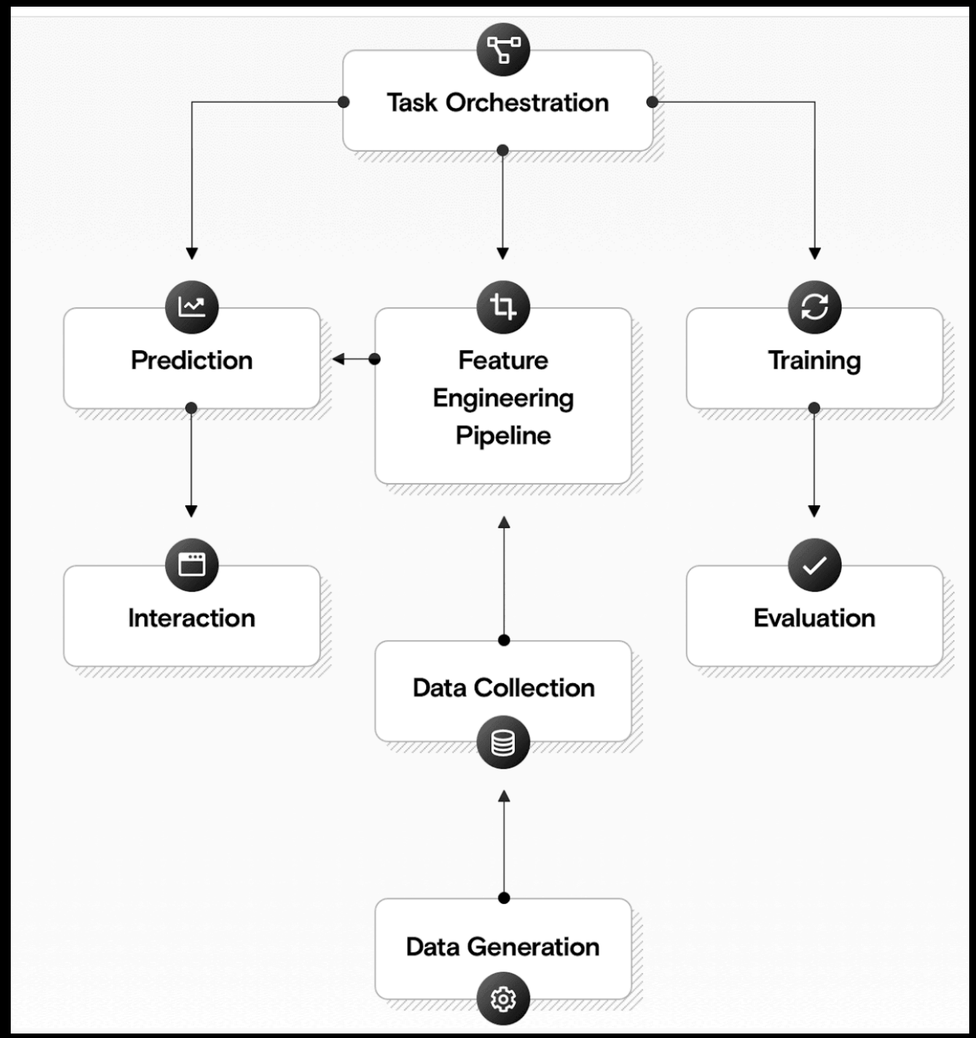
Agenda

- Objective
- ML Solution Architecture
- Data Insight
- Model selection
- Deployment Steps
- Library used in source code
- Source Code

Objective

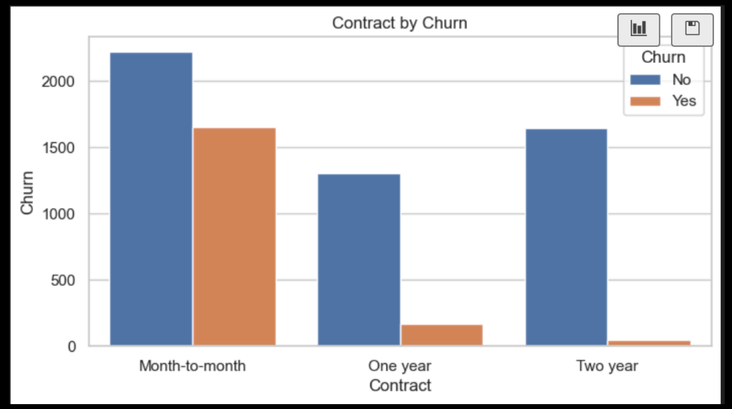
- To analyse the customer churn data and build a machine learning models to predict the customer churn .

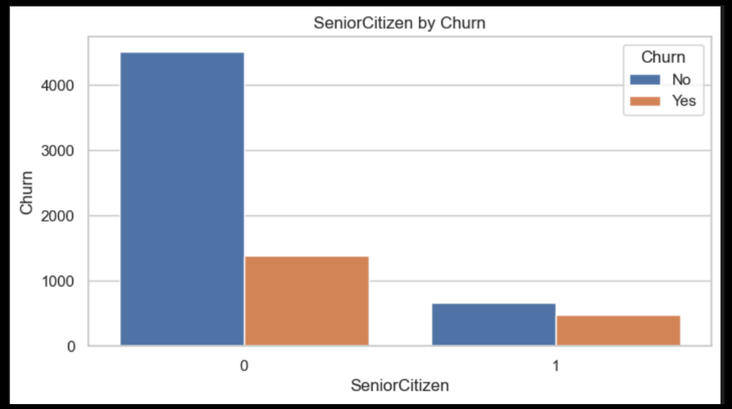
ML Solution Architecture



Data Insight

- Data Shape: (7032, 20)
- Data in class imbalanced: 73%of data of one class.
- Senior Citizen has very high proportion of churn.
- Two year contract has very low churn.





Model Selection

- Label is categorical so model should be classifier.
- Trained two models logistic regression and random forest model.
- Fine Tuned the models with hyper parameters.
- Balanced the data with synthetic data generation of minority class.

Model Evaluation

- Precision and Recall for model evaluation
- Random forest with balanced data worked best:
- Best model scores:
    - Recall : 0.93
    - Precision:0.65
- Random forest model worked well because data has categorical features.
- Random forest models can build non linear boundary.

Deployment Steps

- After training save the model in pickle file
- Upload model in S3 bucket.
- Create a inference script and configuration file.
- Deploy method will give us a end point for inference for real time.

Library used in source code

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
```

Source code for model Training

Logistic regression model Training

[39]:
```
lr = LogisticRegression()
```

[40]:
```
lr.fit(X_train,y_train)
```

[40]:
```
▾ LogisticRegression
LogisticRegression()
```

prediction on test data

[41]:
```
y_pred_lr = lr.predict(X_test)
```

Calculate score

[42]:
```
acc_lr = lr.score(X_test, y_pred_lr)
print("Accuracy_lr", acc_lr)
```

```
Accuracy_lr 1.0
```

Calculate precision and recall

[43]:
```
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
print("Precision_lr:", precision_lr)
print("Recall_lr:", recall_lr)
```

```
Precision_lr: 0.6574074074074074
Recall_lr: 0.56951871657754
```

Fine Tune logistic regression model

[44]:
```
log_reg = LogisticRegression()
```

[45]:
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2'],
              'solver': ['liblinear', 'saga'],
              'class_weight': ['balanced', None]}
grid_search_lr = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid_search_lr.fit(X_train, y_train)
best_params_lr = grid_search_lr.best_params_
```

```
/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge

/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge

/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge

/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge

/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge
```

[46]:
```
best_log_reg = LogisticRegression(**best_params_lr)
best_log_reg.fit(X_train, y_train)
```

```
/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linea

The max_iter was reached which means the coef_ did not converge
```

[46]:
```
▾ LogisticRegression
LogisticRegression(C=10, penalty='l1', solver='saga')
```

Fine tuned logistic regression model evaluation

[47]:
```
y_pred_lr_best = best_log_reg.predict(X_test)
```

Calculate score

[48]:
```
acc_lr_best = best_log_reg.score(X_test, y_test)
print("Accuracy_lr_best", acc_lr_best)
```

```
Accuracy_lr_best 0.8052594171997157
```

Calculate precision and recall

[49]:
```
precision_lr_best = precision_score(y_test, y_pred_lr_best)
recall_lr_best = recall_score(y_test, y_pred_lr_best)
print("Precision_lr_best:", precision_lr_best)
print("Recall_lr_best:", recall_lr_best)
```

```
Precision_lr_best: 0.6524390243902439
Recall_lr_best: 0.5721925133689839
```

Train Random forest model

[50]:
```
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
```

[50]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

prediction on test data

Source code for model Training

prediction on test data

[51]:
```
y_pred_rf = rf_classifier.predict(X_test)
```

Calculate score

[52]:
```
acc_rf = rf_classifier.score(X_test, y_test)
print("Accuracy_fr", acc_rf)
```

```
Accuracy_fr 0.7910447761194029
```

Calculate precision and recall

[53]:
```
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
print("Precision_rf:", precision_rf)
print("Recall_rf:", recall_rf)
```

```
Precision_rf: 0.6282051282051282
Recall_rf: 0.5240641711229946
```

Fine Tune the random forest model

[54]:
```
rf_classifier1 = RandomForestClassifier()
param_grid_rf = {'n_estimators': [100, 200, 300],
                 'max_depth': [None, 10, 20],
                 'min_samples_split': [2, 5, 10]}
grid_search_rf = GridSearchCV(rf_classifier1, param_grid_rf, cv=5, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)
best_params_rf = grid_search_rf.best_params_
best_rf_classifier = RandomForestClassifier(**best_params_rf)
best_rf_classifier.fit(X_train, y_train)
```

[54]:
```
RandomForestClassifier(max_depth=10, min_samples_split=10, n_estimators=300)
```

prediction on test data

[55]:
```
y_pred_rf_best = best_rf_classifier.predict(X_test)
```

Calculate score

[56]:
```
acc_rf_best = best_rf_classifier.score(X_test, y_test)
print("Accuracy_rf_best", acc_rf_best)
```

```
Accuracy_rf_best 0.8031272210376688
```

Calculate precision and recall

[57]:
```
precision_rf_best = precision_score(y_test, y_pred_rf_best)
recall_rf_best = recall_score(y_test, y_pred_rf_best)
print("Precision_rf_best:", precision_rf_best)
print("Recall_rf_best:", recall_rf_best)
```

```
Precision_rf_best: 0.6611295681063123
Recall_rf_best: 0.5320855614973262
```

Source code for model Training

Model training on balanced data

[61]:
```
lr_model_smote = LogisticRegression()
lr_model_smote.fit(X_train_smote, y_train_smote)
```

```
/Users/naina/Desktop/first_project/env/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

[61]:
```
▾ LogisticRegression
LogisticRegression()
```

Model evaluation on smote data

[62]:
```
y_pred_lr_smote = lr_model_smote.predict(X_test)
```

[63]:
```
acc_lr_smote = lr_model_smote.score(X_test, y_test)
print("Accuracy_lr_smote", acc_lr_smote)
```

```
Accuracy_lr_smote 0.7626154939587776
```

Calculate precision and recall

[64]:
```
precision_lr_smote = precision_score(y_test, y_pred_lr_smote)
recall_lr_smote = recall_score(y_test, y_pred_lr_smote)
print("Precision_lr_smote:", precision_lr_smote)
print("Recall_lr_smote:", recall_lr_smote)
```

```
Precision_lr_smote: 0.5367647058823529
Recall_lr_smote: 0.7807486631016043
```

Fine tuned logistic regression model evaluation

[66]:
```
y_pred_lr_best_smote = best_log_reg_smote.predict(X_test)
```

Calculate score

[67]:
```
acc_lr_best_smote = best_log_reg_smote.score(X_test, y_test)
print("Accuracy_lr_best_smote", acc_lr_best_smote)
```

```
Accuracy_lr_best_smote 0.7825159914712153
```

Calculate precision and recall

[68]:
```
precision_lr_best_smote = precision_score(y_test, y_pred_lr_best_smote)
recall_lr_best_smote = recall_score(y_test, y_pred_lr_best_smote)
print("Precision_lr_best_smote:", precision_lr_best_smote)
print("Recall_lr_best_smote:", recall_lr_best_smote)
```

```
Precision_lr_best_smote: 0.567193675889328
Recall_lr_best_smote: 0.767379679144385
```

Train Random forest model on balanced data

[69]:
```
rf_classifier_smote = RandomForestClassifier()
rf_classifier_smote.fit(X_train_smote, y_train_smote)
```

[69]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

prediction on test data

[70]:
```
y_pred_rf_smote = rf_classifier_smote.predict(X_test)
```

Source code for model Training

Calculate score

[71]:
```
acc_rf_smote= rf_classifier_smote.score(X_test, y_test)
print("Accuracy_rf_smote", acc_rf_smote)
```

```
Accuracy_rf_smote 0.837953091684435
```

Calculate precision and recall

[72]:
```
precision_rf_smote = precision_score(y_test, y_pred_rf_smote)
recall_rf_smote = recall_score(y_test, y_pred_rf_smote)
print("Precision_rf_smote:", precision_rf_smote)
print("Recall_rf_smote:", recall_rf_smote)
```

```
Precision_rf_smote: 0.631768953068592
Recall_rf_smote: 0.9358288770053476
```

Fine Tune the random forest model on balanced data

[73]:
```
rf_classifier2 = RandomForestClassifier()
param_grid_rf_smote = {'n_estimators': [100, 200, 300],
              'max_depth': [None, 10, 20],
              'min_samples_split': [2, 5, 10]}
grid_search_rf_smote = GridSearchCV(rf_classifier2, param_grid_rf_smote, cv=5, scoring='accuracy')
grid_search_rf_smote.fit(X_train_smote, y_train_smote)
best_params_rf_smote =grid_search_rf_smote.best_params_
best_rf_classifier_smote = RandomForestClassifier(**best_params_rf_smote)
best_rf_classifier_smote.fit(X_train_smote, y_train_smote)
```

[73]:
```
RandomForestClassifier(max_depth=20, n_estimators=200)
```

prediction on test data

[74]:
```
y_pred_rf_best_smote= best_rf_classifier_smote.predict(X_test)
```

```
grid_search_rf_smote.fit(X_train_smote, y_train_smote)
best_params_rf_smote =grid_search_rf_smote.best_params_
best_rf_classifier_smote = RandomForestClassifier(**best_params_rf_smote)
best_rf_classifier_smote.fit(X_train_smote, y_train_smote)
```

[73]:
```
RandomForestClassifier(max_depth=20, n_estimators=200)
```

prediction on test data

[74]:
```
y_pred_rf_best_smote= best_rf_classifier_smote.predict(X_test)
```

Calculate score

[75]:
```
acc_rf_best_smote = best_rf_classifier_smote.score(X_test_smote, y_test_smote)
print("Accuracy_rf_best_smote", acc_rf_best_smote)
```

```
Accuracy_rf_best_smote 0.8451113262342691
```

Calculate precision and recall

[76]:
```
precision_rf_best_smote = precision_score(y_test, y_pred_rf_best_smote)
recall_rf_best_smote = recall_score(y_test, y_pred_rf_best_smote)
print("Precision_rf_best_smote:", precision_rf_best_smote)
print("Recall_rf_best_smote:", recall_rf_best_smote)
```

```
Precision_rf_best_smote: 0.6345454545454545
Recall_rf_best_smote: 0.9331550802139037
```

[ ]:

Thank You