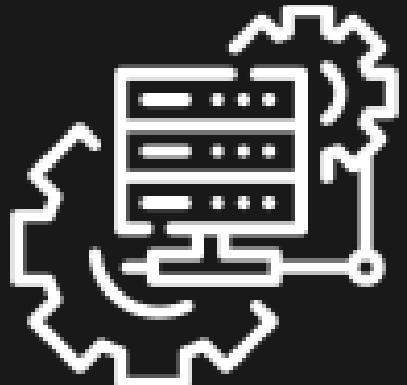


은행 신용카드 신규 고객의

카드 해지 점수 예측 및 마케팅 MLOPS 시스템



데이터애널리틱스 특수연구 III



팀명 : 5팀

2024020432 고태영

2024020431 곽동호

2020150462 이소희

2024020428 이충은

2024020436 최민경

프로젝트 주제 및 목표

“MLOPS를 활용한 은행의 신용카드 사업 수입 극대화”

✓ 문제 원천 봉쇄

고객의 한달 후 카드 해지 가능성 예측을 통한 선천적 문제 해결

✓ 문제 발생 시 빠른 대처 가능

일정량의 데이터가 쌓이면 모델 지속적 업데이트

- 1 고객의 카드 해지율 감소
- 2 마케팅 비용 감소
- 3 고객 만족도 증가
- 4 시스템 자원의 효율성 증대

데이터 소개

신용카드 보유 고객 개인정보 및 카드 해지 여부에 대한 데이터

<https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers>

총데이터 개수
10,127

Feature 개수
21

CLIENTNUM	고객 고유 식별 번호	Income_Category	소득 수준	Total_Revolving_Bal	총 리볼빙 잔고
Attrition_Flag	이탈 여부 [종속변수: 한 달 뒤 이탈 여부]	Card_Category	카드 종류	Avg_Open_To_Buy	평균 사용 가능 금액
Customer_Age	고객 나이	Months_on_book	은행 가입 기간	Total_Amt_Chng_Q4_Q1	4분기에서 1분기까지 의 거래 변화 비율
Gender	성별	Total_Relationship_Count	가입한 상품 수	Total_Trans_Amt	총 거래 금액
Dependent_count	부양 가족 수	Months_Inactive_12_mon	12개월 동안 비활성화된 개월 수	Total_Trans_Ct	총 거래 횟수
Education_level	교육 수준	Contacts_Count_12_mon	12개월 동안 고객과의 연락 횟수	Total_Ct_Chng_Q4_Q1	4분기에서 1분기까지 총 거래 횟수 변화 비율
Marital_Status	결혼 여부	Credit_Limit	신용 한도	Avg_Utilization_Ratio	평균 카드 이용 비율

EDA

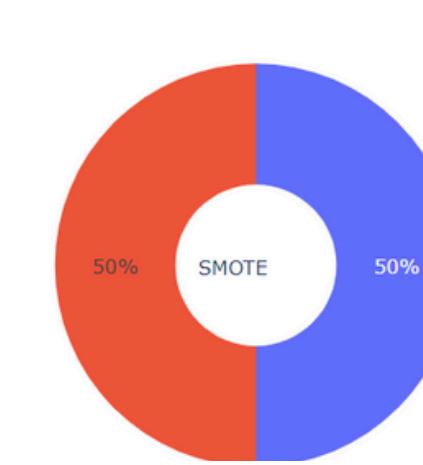
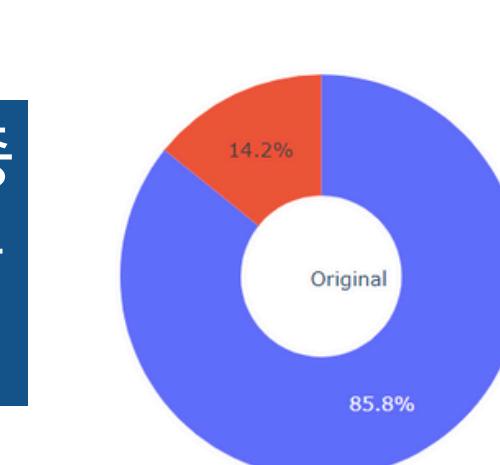
02

실제 값을 모르는 *Categorical* 변수인 경우
*unknown*이라는 하나의 새로운 그룹
생성

01

0.8보다 높은 *corr*를 보이는 '12개월 평균
카드한도과 계좌금액의 차이' 변수 제거

Class distribution

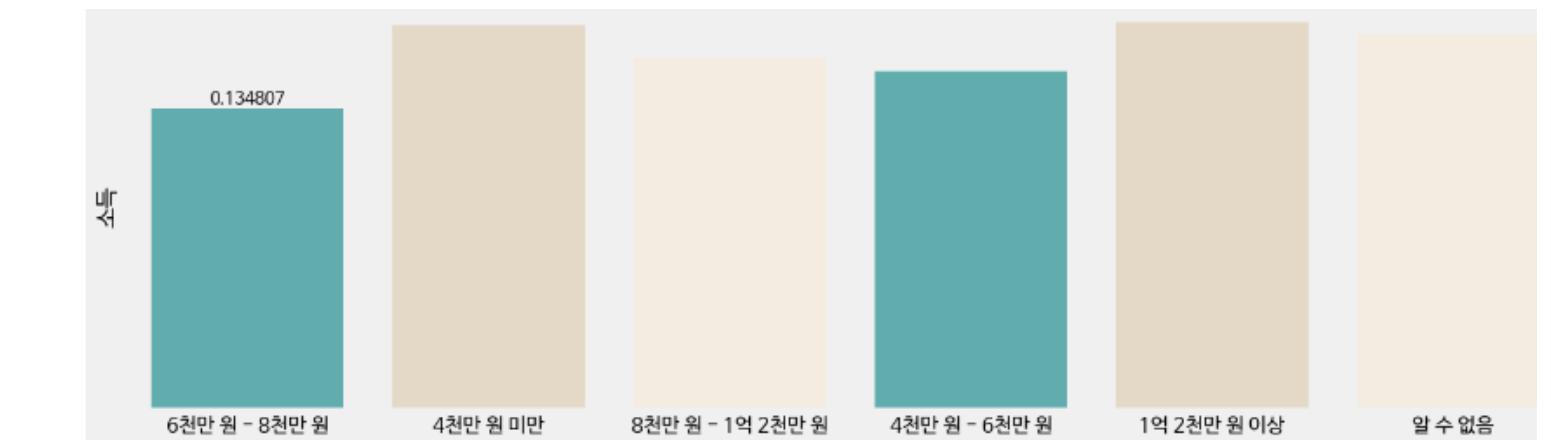


0

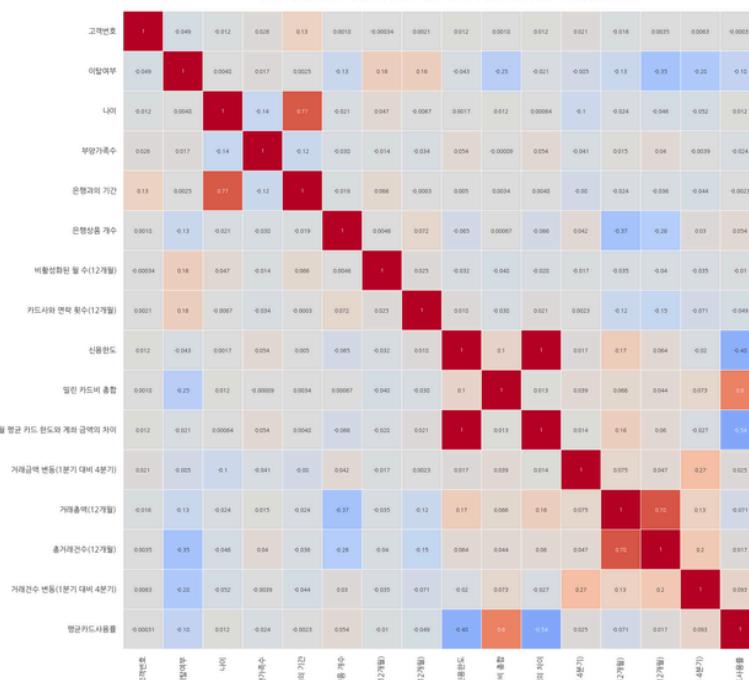
1

03

SMOTE 를 활용하여 불균형한 데이터셋(종
속 변수 기준)에서 소수 클래스의 데이터를
증강

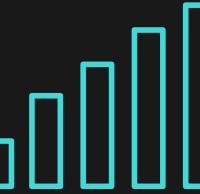


Correlation between all the Numerical Features



-1.0
-0.8
-0.6
-0.4
-0.2
0.0
0.2
0.4
0.6
0.8
1.0

문제 상황 및 해결 방안



기존 및 신규 고객의 신용카드 해지 예측의 부정확성

해결 방안 : 해지 가능성(점수제) 예측,
변수별 특성을 반영한 여러 모델 구축

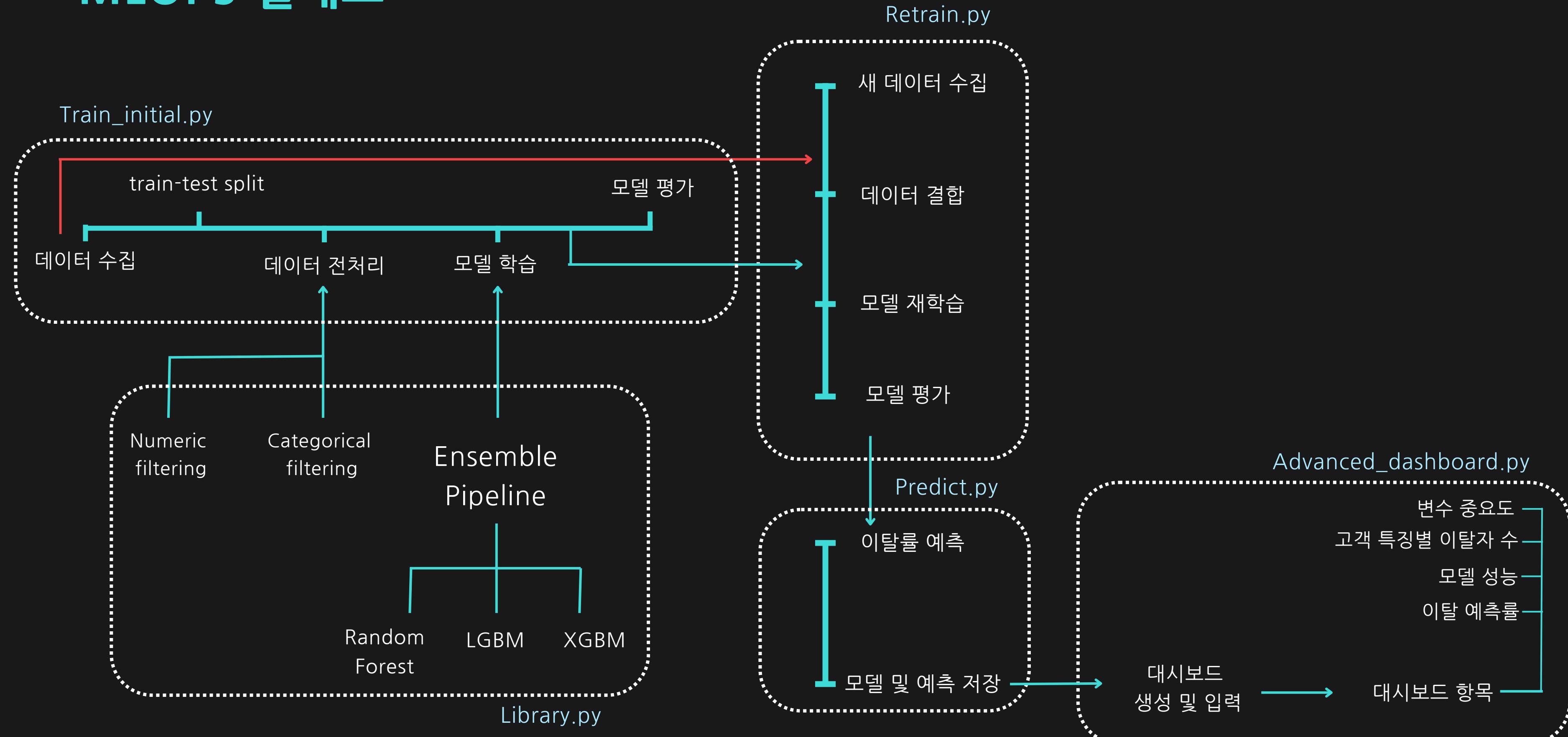
이탈률 감소를 위한 전략의 구체성 및 사전 대책 부족

해결 방안 : 실시간 대응 시스템 구축,
개인별 맞춤 마케팅 전략 사용

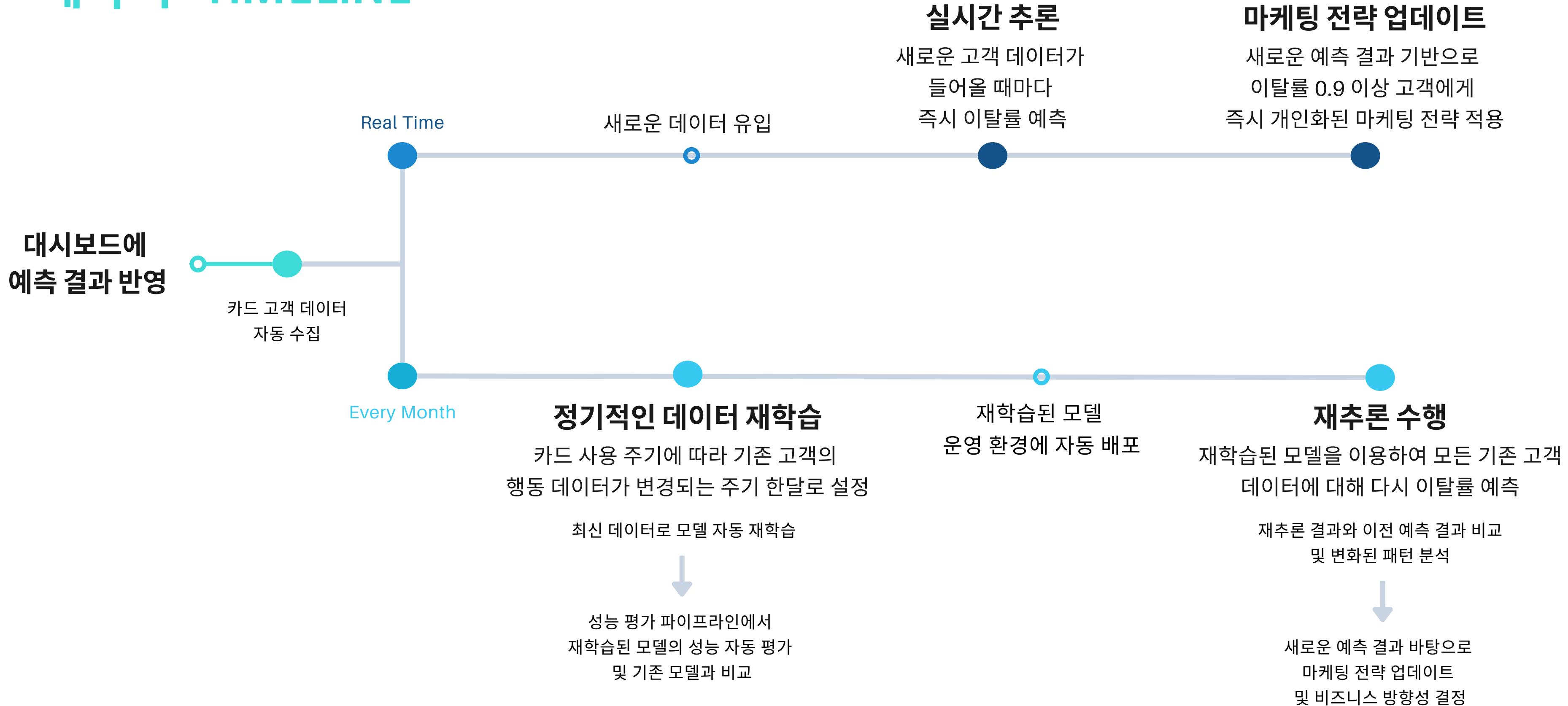
데이터 처리 및 모델 업데이트의 비효율성 및 어려움

해결 방안 : 파이프라인 자동화를 통한 MLOPS 시스템 구축,
대시보드를 통한 관리자의 모니터링,
데이터 및 모델 관련 각종 상황 발생에 대한 실시간 알림

MLOPS 설계도



예측 후 TIMELINE



코드 설명

Train_initial.py

```
● ● ●  
data = pd.read_csv('고객이탈률데이터.csv') → 데이터 수집  
  
data['이탈여부'] = data['이탈여부'].map({'기존고객': 0, '이탈고객': 1})  
  
X = data.drop(columns=['이탈여부'])  
y = data['이탈여부'] → 종속변수(한 달 뒤 이탈 여부)  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) → Train-test split  
  
original_data = {  
    'X_train': X_train,  
    'y_train': y_train  
}  
joblib.dump(original_data, 'original_training_data.pkl')  
  
ensemble_model = EnsemblePipeline()  
ensemble_model.fit(X_train, y_train) → 모델 학습  
  
y_train_pred = ensemble_model.predict(X_train)  
print("Training data evaluation")  
print(classification_report(y_train, y_train_pred['ensemble'].astype(int)))  
  
y_test_pred = ensemble_model.predict(X_test) → 모델 평가  
print("Test data evaluation")  
print(classification_report(y_test, y_test_pred['ensemble'].astype(int)))  
  
joblib.dump(ensemble_model, 'initial_model.pkl')
```

코드 설명 Library.py

```
● ● ●  
  
class CorrelationFilter(BaseEstimator, TransformerMixin):  
    def __init__(self, threshold=0.8):  
        self.threshold = threshold  
        self.final_cols = None  
  
    def fit(self, X, y=None):  
        X_df = pd.DataFrame(X)  
        corr_matrix = X_df.corr().abs()  
        upper_tri = corr_matrix.where(np.triu(np.ones((corr_matrix.shape), k=1)).astype(np.bool_))  
        to_drop = [column for column in upper_tri.columns  
                  if any(upper_tri[column] > self.threshold)]  
        self.final_cols = [col for col in X_df.columns  
                           if col not in to_drop]  
        return self  
  
    def transform(self, X):  
        X_df = pd.DataFrame(X)  
        return X_df[self.final_cols]
```

Correlation Filter: 상관성 높은 변수 제거

Numeric Filtering : 범주형 변수 전처리

Categorical Filtering : 연속형 변수 전처리

```
● ● ●  
  
class NumericFiltering(BaseEstimator, TransformerMixin):  
    def __init__(self, check_const_col=True, check_id_col=True):  
        self.check_const_col = check_const_col  
        self.check_id_col = check_id_col  
  
    def fit(self, X, y=None):  
        X = np.array(X)  
        if self.check_const_col:  
            self.constant_col = [i for i in range(X.shape[1])  
                                 if X[:, i].std() == 0]  
        else:  
            self.constant_col = []  
  
        if self.check_id_col:  
            self.id_col = [i for i in range(X.shape[1])  
                          if len(np.unique(np.diff(X[:, i]))) == 1]  
        else:  
            self.id_col = []  
  
        self.rm_cols = self.constant_col + self.id_col  
        self.final_cols = [i for i in range(X.shape[1])  
                           if i not in self.rm_cols]  
        return self  
  
    def transform(self, X):  
        X = np.array(X)  
        result = X[:, self.final_cols]  
        return result
```

```
● ● ●  
  
class CategoricalFiltering(BaseEstimator, TransformerMixin):  
    def __init__(self, check_const_col=True, check_id_col=True, check_cardinality=True):  
        self.check_const_col = check_const_col  
        self.check_id_col = check_id_col  
        self.check_cardinality = check_cardinality  
  
    def fit(self, X, y=None):  
        X = np.array(X)  
        if self.check_const_col:  
            self.constant_col = [i for i in range(X.shape[1])  
                                 if len(np.unique(X[:, i])) == 1]  
        else:  
            self.constant_col = []  
  
        if self.check_id_col:  
            self.id_col = [i for i in range(X.shape[1])  
                          if len(np.unique(X[:, i])) == X.shape[0]]  
        else:  
            self.id_col = []  
  
        if self.check_cardinality:  
            self.cardinality = [i for i in range(X.shape[1])  
                                if len(np.unique(X[:, i])) > 50]  
        else:  
            self.cardinality = []  
  
        self.rm_cols = self.constant_col + self.id_col + self.cardinality  
        self.final_cols = [i for i in range(X.shape[1]) if i not in self.rm_cols]  
        return self  
  
    def transform(self, X):  
        X = np.array(X)  
        result = X[:, self.final_cols]  
        return result
```

코드 설명

Library.py

```
class EnsemblePipeline:  
    def __init__(self):  
        pipe1 = Pipeline([  
            ('step1', SimpleImputer(strategy="mean")),  
            ('step2', NumericFiltering()),  
            ('step3', StandardScaler()),  
            ('step4', CorrelationFilter(threshold=0.8)),  
        ])  
  
        pipe2 = Pipeline([  
            ('step1', SimpleImputer(strategy="most_frequent")),  
            ('step2', CategoricalFiltering()),  
            ('step3', OneHotEncoder()),  
        ])  
  
        transform = ColumnTransformer([  
            ('num', pipe1, make_column_selector(dtype_include=np.number)),  
            ('cat', pipe2, make_column_selector(dtype_exclude=np.number)),  
        ])  
  
        self.models = {  
            'RF': RandomForestClassifier(),  
            'LGBM': LGBMClassifier(),  
            'XGBM': XGBClassifier()  
        }  
  
        ensemble_models = []  
        for model in self.models.keys():  
            pipe0 = ImbPipeline([  
                ('transform', transform),  
                ('smote', SMOTE()),  
                ('model', self.models[model])  
            ])  
            ensemble_models.append((model, pipe0))  
  
        self.ensemble_pipe = StackingClassifier(estimators=ensemble_models,  
                                                final_estimator=XGBClassifier())  
  
    def fit(self, X, y):  
        self.columns = X.columns.tolist()  
        self.ensemble_pipe.fit(X, y)  
        self.feature_names = self._get_final_feature_names(X)  
        self.imp = permutation_importance(estimator=self.ensemble_pipe,  
                                           X=X, y=y,  
                                           scoring="accuracy",  
                                           n_repeats=5)  
        self.feature_importances_ = pd.DataFrame(self.imp['importances_mean'], index=self.columns,  
                                                columns=['importance']).sort_values('importance', ascending=False)  
        return self  
  
    def predict(self, X):  
        pred = pd.DataFrame()  
        for model in self.models.keys():  
            pred[model] = self.ensemble_pipe.named_estimators_[model].predict(X)  
        # 각 분류기의 예측값을 평균하여 양상을 예측값 반환  
        pred['ensemble'] = pred.mean(axis=1)  
        return pred  
  
    def predict_proba(self, X):  
        pred = pd.DataFrame(self.ensemble_pipe.predict_proba(X[:, 1], columns=['ensemble'])  
        for model in self.models.keys():  
            pred[model] = self.ensemble_pipe.named_estimators_[model].predict_proba(X[:, 1])  
        return pred  
  
    def feature_importances(self):  
        return self.feature_importances_  
  
    def get_feature_names(self):  
        return self.feature_names
```

각 변수 표준화

Ensemble 모델
(RandomForest,
LGBM, XGBM)

Pipeline

Column Transformer

Pipeline 1

Simple Imputer

Numeric Filtering

StandardScaler

Correlation Filter

Pipeline 2

Simple Imputer

Categorical Filtering

OneHot Encoder

Correlation Filter

SMOTE

Ensemble Model
(RandomForest, LGBM, XGBM)

코드 설명
Retrain.py

```
original_data = joblib.load('original_training_data.pkl')
X_train_original = original_data['X_train']
y_train_original = original_data['y_train']
```

```
data = pd.read_csv('New_고객이탈률데이터.csv')
```

```
client_nums = data['고객번호']
genders = data['성별']
income_categories = data['소득']
education_levels = data['교육수준']
marital_statuses = data['결혼상태']
```

```
data['이탈여부'] = data['이탈여부'].map({'기존고객': 0, '이탈고객': 1})
X_new_data = data.drop(columns=['이탈여부', '고객번호'])
y_new_data = data['이탈여부']
```

```
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new_data, y_new_data,
test_size=0.2, random_state=42)
```

```
X_train_combined = pd.concat([X_train_original, X_train_new], axis=0)
y_train_combined = pd.concat([y_train_original, y_train_new], axis=0)
```

```
ensemble_model = joblib.load('initial_model.pkl')
ensemble_model.fit(X_train_combined, y_train_combined)
```

```
y_train_combined_pred = ensemble_model.predict(X_train_combined)
model_performance = classification_report(y_train_combined,
y_train_combined_pred['ensemble'].astype(int), output_dict=True)
print("Retrained model evaluation on combined training data")
print(classification_report(y_train_combined, y_train_combined_pred['ensemble'].astype(int),
output_dict=True))
```

```
joblib.dump(ensemble_model, 'retrained_model.pkl')
```

새로운 데이터 수집

기존 데이터와 새로운 데이터
합쳐서 모델 재학습 및 평가

재학습된 모델 저장

코드 설명

Predict.py

```
ensemble_model = joblib.load('retrained_model.pkl')      ----- 재학습된 모델 불러오기  
y_test_new_pred = ensemble_model.predict_proba(X_test_new)[‘ensemble’] ----- 새로운 데이터에 대한 예측 진행  
  
predictions = pd.DataFrame({  
    ‘고객번호’: client_nums.iloc[X_test_new.index].values,  
    ‘성별’: genders.iloc[X_test_new.index].values,  
    ‘소득’: income_categories.iloc[X_test_new.index].values,  
    ‘교육수준’: education_levels.iloc[X_test_new.index].values,  
    ‘결혼상태’: marital_statuses.iloc[X_test_new.index].values,  
    ‘예측이탈률’: y_test_new_pred,  
    ‘예측이탈여부’: y_test_new.values  
})  
top_10_churn = predictions.nlargest(10, ‘예측이탈률’)
```

```
predictions.to_csv(‘new_data_predictions.csv’, index=False)  
top_10_churn.to_csv(‘top_10_churn.csv’, index=False)  
joblib.dump(model_performance, ‘model_performance.pkl’)
```

대시보드 반영 위해
예측값 저장

코드 설명 Advanced_dashboard.py

● ● ●

```
predictions = pd.read_csv('new_data_predictions.csv')
top_10_churn = predictions.nlargest(10, '예측이탈률')
model_performance = joblib.load('model_performance.pkl')

ensemble_model = joblib.load('retrained_model.pkl')
feature_importances = ensemble_model.feature_importances().reset_index().rename(columns={'index': '변수', 'importance': '중요도'})

accuracy = model_performance['accuracy']
precision = model_performance['weighted avg']['precision']
recall = model_performance['weighted avg']['recall']
f1_score = model_performance['weighted avg']['f1-score']

app = Dash(__name__)
app.layout = html.Div([
    html.H1("고객 이탈 예측 대시보드"),
    html.H2("예측 이탈률 상위 10명 고객"),
    html.Table([
        html.Thead(html.Tr([html.Th(col) for col in top_10_churn.columns])),
        html.Tbody([
            html.Tr([
                html.Td(top_10_churn.iloc[i][col], style={'color': 'blue'} if i < 10 else {}),
                for col in top_10_churn.columns
            ]) for i in range(len(top_10_churn))
        ])
    ]),
    html.H2("모델 성능"),
    html.Table([
        html.Tr([html.Th('평가 지표'), html.Th('값')]),
        html.Tr([html.Td('정확도'), html.Td(f"{accuracy:.2f}")]),
        html.Tr([html.Td('정밀도'), html.Td(f"{precision:.2f}")]),
        html.Tr([html.Td('재현율'), html.Td(f"{recall:.2f}")]),
        html.Tr([html.Td('F1 점수'), html.Td(f"{f1_score:.2f}")]),
    ]),
])
```

```
# Feature importances
html.H2("변수 중요도"),
dcc.Graph(id='feature-importances-bar', figure=px.bar(feature_importances, x='변수', y='중요도',
title='변수 중요도')),

# Dropdowns for filtering
html.Div([
    html.Div([
        html.Label('성별 필터'),
        dcc.Dropdown(
            id='gender-filter',
            options=[{'label': gender, 'value': gender} for gender in predictions['성별'].unique()],
            value=None,
            multi=True,
            placeholder="성별 선택"
        ),
    ], style={'width': '30%', 'display': 'inline-block'}),
    html.Div([
        html.Label('소득 수준'),
        dcc.Dropdown(
            id='income-filter',
            options=[{'label': income, 'value': income} for income in predictions['소득'].unique()],
            value=None,
            multi=True,
            placeholder="소득 수준 선택"
        ),
    ], style={'width': '30%', 'display': 'inline-block', 'marginLeft': '20px'}),
    html.Div([
        html.Label('교육 수준'),
        dcc.Dropdown(
            id='education-filter',
            options=[{'label': edu, 'value': edu} for edu in predictions['교육수준'].unique()],
            value=None,
            multi=True,
            placeholder="교육 수준 선택"
        ),
    ], style={'width': '30%', 'display': 'inline-block', 'marginLeft': '20px'}),
    html.Div([
        html.Label('결혼 여부'),
        dcc.Dropdown(
            id='marital-filter',
            options=[{'label': marital, 'value': marital} for marital in predictions['결혼상태'].unique()],
            value=None,
            multi=True,
            placeholder="결혼 여부 선택"
        ),
    ], style={'width': '30%', 'display': 'inline-block', 'marginLeft': '20px'}),
], style={'marginBottom': '20px'})
```

코드 설명 Advanced_dashboard.py

```
# Histogram of churn probabilities
html.H2("이탈률 분포"),
dcc.Graph(id='churn-histogram'),

# Bar chart of churn counts by gender
html.H2("성별에 따른 이탈자 수"),
dcc.Graph(id='churn-gender-bar'),

# Bar chart of churn counts by income category
html.H2("소득 수준에 따른 이탈자 수"),
dcc.Graph(id='churn-income-bar'),

# Bar chart of churn counts by education level
html.H2("교육 수준에 따른 이탈자 수"),
dcc.Graph(id='churn-education-bar'),

# Bar chart of churn counts by marital status
html.H2("결혼 여부에 따른 이탈자 수"),
dcc.Graph(id='churn-marital-bar'),

# Display a table with the predictions
html.H2("모든 고객 예측표"),
dash_table.DataTable(
    id='churn-table',
    columns=[{'name': col, 'id': col} for col in predictions.columns],
    data=predictions.to_dict('records'),
    style_data_conditional=[
        {
            'if': {
                'filter_query': '{예측이탈률} > 0.9',
                'column_id': '예측이탈률'
            },
            'backgroundColor': 'tomato',
            'color': 'white'
        }
    ],
    sort_action='native',
    filter_action='native',
    page_size=10
)

# Callback to update the visualizations based on the selected filters
@app.callback(
    [Output('churn-histogram', 'figure'),
     Output('churn-gender-bar', 'figure'),
     Output('churn-income-bar', 'figure'),
     Output('churn-education-bar', 'figure'),
     Output('churn-marital-bar', 'figure'),
     Output('churn-table', 'data')],
    [Input('gender-filter', 'value'),
     Input('income-filter', 'value'),
     Input('education-filter', 'value'),
     Input('marital-filter', 'value')]
)
```

```
def update_graphs(selected_genders, selected_income_categories, selected_educations,
                  selected_maritals):
    filtered_predictions = predictions.copy()

    if selected_genders:
        filtered_predictions = filtered_predictions[filtered_predictions['성별'].isin(selected_genders)]
    if selected_income_categories:
        filtered_predictions = filtered_predictions[filtered_predictions['소득'].isin(selected_income_categories)]
    if selected_educations:
        filtered_predictions = filtered_predictions[filtered_predictions['교육수준'].isin(selected_educations)]
    if selected_maritals:
        filtered_predictions = filtered_predictions[filtered_predictions['결혼상태'].isin(selected_maritals)]

    # Histogram of churn probabilities
    churn_histogram = px.histogram(
        filtered_predictions,
        x='예측이탈률',
        nbins=50,
        title='이탈률 분포'
    )

    # Bar chart of churn counts by gender
    churn_gender_bar = px.bar(
        filtered_predictions.groupby('성별').size().reset_index(name='Counts'),
        x='성별',
        y='Counts',
        title='성별에 따른 이탈자 수'
    )

    # Bar chart of churn counts by income category
    churn_income_bar = px.bar(
        filtered_predictions.groupby('소득').size().reset_index(name='Counts'),
        x='소득',
        y='Counts',
        title='소득 수준에 따른 이탈자 수'
    )

    # Bar chart of churn counts by education level
    churn_education_bar = px.bar(
        filtered_predictions.groupby('교육수준').size().reset_index(name='Counts'),
        x='교육수준',
        y='Counts',
        title='교육 수준에 따른 이탈자 수'
    )

    # Bar chart of churn counts by marital status
    churn_marital_bar = px.bar(
        filtered_predictions.groupby('결혼상태').size().reset_index(name='Counts'),
        x='결혼상태',
        y='Counts',
        title='결혼 여부에 따른 이탈자 수'
    )

    return churn_histogram, churn_gender_bar, churn_income_bar, churn_education_bar, churn_marital_bar,
           filtered_predictions.to_dict('records')

if __name__ == '__main__':
    port = int(os.environ.get('PORT', 8050))
    app.run_server(debug=True, port=port)
```

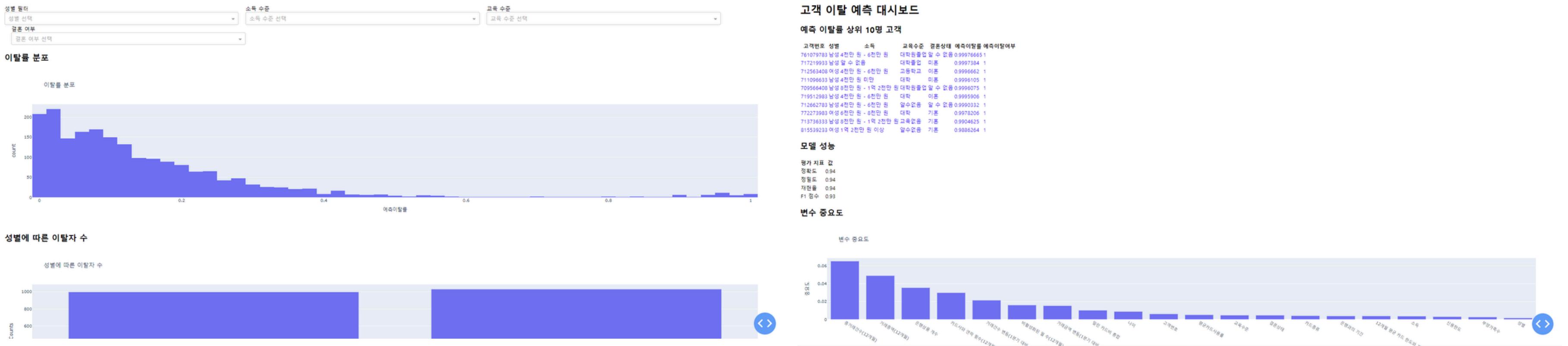
Dashboard 사이트 로컬 배포

<https://horang2.loca.lt/>

모든 고객 예측표

#	고객번호	성별	소득	교육수준	결혼상태	예측이탈률	예측이탈여부
1	filter data...						
2	803335008	여성	할 수 있음	대학	할 수 있음	0.4732755	0
3	721448283	남성	6천만 원 ~ 8천만 원	대학졸업	기혼	0.07188848	1
4	713768358	남성	할 수 있음	대학졸업	이혼	0.06975392	0
5	713834358	남성	4천만 원 ~ 6천만 원	대학졸업	이혼	0.17573287	0
6	708708858	여성	4천만 원 미만	고등학교	이혼	0.93097665	1
7	721508283	여성	할 수 있음	할수없음	미혼	0.013124279	0
8	788658483	여성	6천만 원 ~ 8천만 원	대학	미혼	0.24064186	0
9	709235683	남성	4천만 원 미만	대학	미혼	0.0246558827	0
10	714603558	남성	6천만 원 ~ 8천만 원	고등학교	기혼	0.090724945	0
11	717622308	남성	할 수 있음	대학	이혼	0.2083788	0

Score가 0.9 이상인 고객이 강조되어 표시된다.



마케팅 전략

1

마케팅 대상자 파악

- Score 기준으로 한 달 뒤 해지 가능성 상위, 하위 10% 고객 파악
- SHAP, PDP, ICE plot을 이용하여 특정 변수가 예측에 미치는 영향 파악 및 시각화

2

군집화

- 마케팅 대상자 파악 후 K-means, DBSCAN 등 군집화 알고리즘을 적용하여 고객을 분류
- 각 군집의 특성을 파악하여 맞춤형 마케팅 전략 수립 가능

3

맞춤형 마케팅 전략

- 상위 10% 고객
 - 각 고객의 해지 가능성에 영향을 미치는 주요 변수를 파악 후 이를 기반으로 맞춤형 혜택 제공
 - 충성도가 낮은 고객에게 포인트 적립, 특별 할인 쿠폰, 전용 이벤트 초대 등 제공
 - 고객 상담 서비스를 개선하여 주요 불만 요소 파악 후 신속한 해결
- 하위 10% 고객
 - 장기 고객에게 감사의 의미로 특별 혜택 제공
 - 만족도가 높은 고객의 피드백을 수집하고 이를 서비스 개선에 반영하여 지속적인 만족도 유지

결론 및 제언

- 은행의 신용카드 해지율 예측 정확도 향상
 - 파이프라인 자동화를 통해 데이터 처리 및 모델 업데이트의 효율성 향상
 - 대시보드를 통해 관리자의 실시간 모니터링 가능
 - 군집과 개개인의 특성에 맞는 마케팅 전략 수립
-
- 모델의 예측 성능을 주기적으로 평가, 최신 데이터를 반영하여 지속적으로 개선
 - 새로운 데이터를 실시간으로 반영할 수 있는 자동화된 시스템을 구축
 - 대시보드로 실시간 모델 성능, 데이터 처리 상황 모니터링 및 실시간 알림 시스템을 구축하여 이슈 발생 시 신속한 대응 가능
 - 일관되고 정확한 데이터 분석을 위해 다양한 채널에서 수집되는 고객 데이터 통합 관리 및 데이터 품질관리 시스템 필요

