Airline Passenger Satisfaction

Context

This dataset contains an airline passenger satisfaction survey. What factors are highly correlated to a satisfied (or dissatisfied) passenger? Can you predict passenger satisfaction?

Content

Gender: Gender of the passengers (Female, Male)

Customer Type: The customer type (Loyal customer, disloyal customer)

Age: The actual age of the passengers

Type of Travel: Purpose of the flight of the passengers (Personal Travel, Business Travel)

Class: Travel class in the plane of the passengers (Business, Eco, Eco Plus)

Flight distance: The flight distance of this journey

Inflight wifi service: Satisfaction level of the inflight wifi service (0:Not Applicable;1-5)

Departure/Arrival time convenient: Satisfaction level of Departure/Arrival time convenient

Ease of Online booking: Satisfaction level of online booking

Gate location: Satisfaction level of Gate location

Food and drink: Satisfaction level of Food and drink

Online boarding: Satisfaction level of online boarding

Seat comfort: Satisfaction level of Seat comfort

Inflight entertainment: Satisfaction level of inflight entertainment

On-board service: Satisfaction level of On-board service

Leg room service: Satisfaction level of Leg room service

Baggage handling: Satisfaction level of baggage handling

Check-in service: Satisfaction level of Check-in service

Inflight service: Satisfaction level of inflight service

Cleanliness: Satisfaction level of Cleanliness

Departure Delay in Minutes: Minutes delayed when departure

Arrival Delay in Minutes: Minutes delayed when Arrival

Satisfaction: Airline satisfaction level(Satisfaction, neutral or dissatisfaction)

데이터 전처리

1.결측치 처리

- 데이터 결측치 확인 결과 : ‘Arrival Delay in Minutes’ 에서 결측치 확인
- ‘Arrival Delay in Minutes’ 에 따른 satisfaction 빈도 분석 한 결과 약 6:4(불만족: 만족)로 의미가 있는 데이터로 추정
- 일반적으로 결측치를 채울 때 빈도 수가 높은 것, 중앙값, 평균으로 채울 수 있지만 다른 방법 고려함
- K-최근접 이웃(K-Nearest Neighbors, KNN) 알고리즘: 유사한 기록을 가진 다른 데이터들을 찾아서, 그 유사한 데이터들의 'Arrival Delay in Minutes' 값을 평균내어 누락된 값을 추정
- KNN 알고리즘을 사용하여 결측치 채움

2.중복값 확인

- 중복값 확인 결과 중복값 없음

3.데이터 타입 변경

- 데이터 타입과 데이터 수치 확인 결과 object, int, float 3개의 타입이 있음
- 모든 수치형 데이터는 정수형 타입으로 되어있기에 int 타입으로 변경

4. 불필요한 변수 삭제 및 변수 명 변경

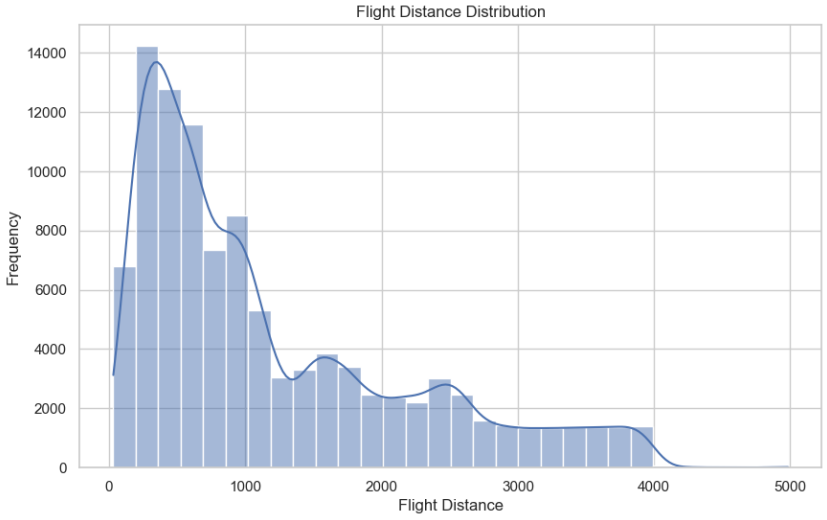
- ‘Unnamed: 0’ , ‘id’ 는 모델링 할 때 필요 없으므로 삭제
- 변수명들 좀 더 알아보기 쉽고 띄어쓰기와 대문자 없이 변경

5. 범주형 데이터 처리

- 범주형 데이터 확인 결과 ‘class', 'gender', 'customer_tyoe', 'tyoe_travel', 'satisfaction’ 인코딩 필요
- 일반적인 인코딩 기법 3가지 순서형, 레이블, 원-핫 인코딩이 있음
- 순서형 데이터는 순서 및 등급이 있을 때 사용하기 유용하며, 레이블과 원-핫은 범주 간 순서가 중요하지 않을 때 사용한다. 원-핫 인코딩을 사용하지 않은 이유는, 사용 시 열 개수가 늘어나 데이터 차원의 증가가 혹여나 모델링에 끼치는 영향을 고려
- ‘class’ 등급으로 나뉘므로 순서형 인코딩 적용
- ‘'gender', 'customer_type', 'type_travel','satisfaction’ 레이블 인코딩 적용

6.이상치 처리

- test 데이터의 이상치는 건들지 않고 train만 이상치 처리
- z-score 가 3 이상인 데이터들 확인 결과 : age, flight_distance, inflight_service, departure_delay, arrival_delay 이상치 확인됨
- ‘age’ 같은 경우 데이터 확인해 봤을 때 7-85세로 구분되어있어 이상치 처리 할 필요 없음
- ‘flight_distance’ 같은 경우 z-score가 3 이상인 데이터들은 4230 이상인 데이터들이며 약 58개 있음



- 이상치를 제거하기보단 범주화 시도

- 단 거리 비행 (Short-haul): 보통 500마일(약 800킬로미터) 이하의 비행 → 1
  중 거리 비행 (Medium-haul): 대략 500-1500마일(약 800-2400킬로미터) 사이의 비행 → 2
  장 거리 비행 (Long-haul): 일반적으로 1500마일(약 2400킬로미터) 이상의 비행 → 3

- **Training Dataset:**

  Flight distance category 1 (≤ 500 miles): 32,321

  Flight distance category 2 (> 500 and ≤ 1500 miles): 40,112

  Flight distance category 3 (> 1500 miles): 31,471

- **Testing Dataset:**

  Flight distance category 1 (≤ 500 miles): 7,955

  Flight distance category 2 (> 500 and ≤ 1500 miles): 10,140

  Flight distance category 3 (> 1500 miles): 7,881

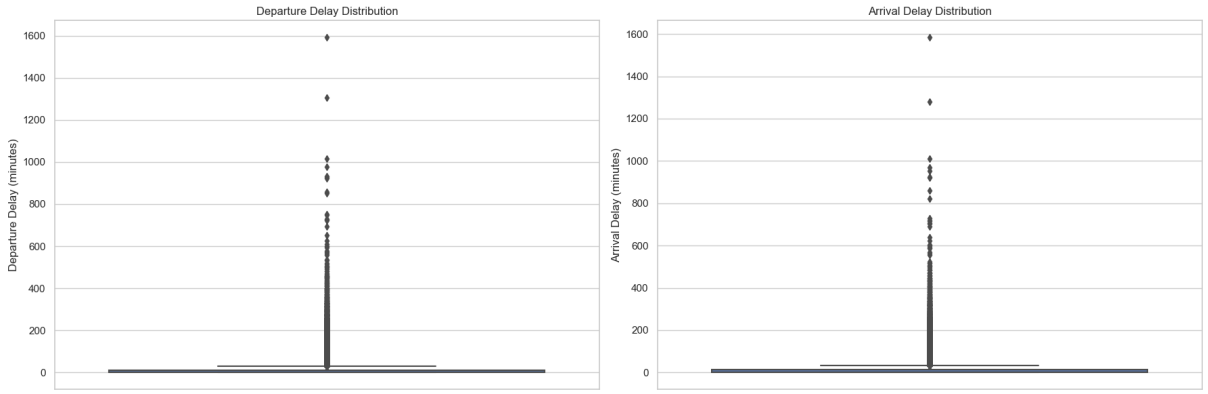
- 균형 잡힌 분포를 가지므로 이상치 처리보단 범주화 하는 것으로 결정

- ‘inflight_service’ 확인 결과 3개의 데이터가 0으로 나타나지만, 원래 1, 2, 3, 4, 5의 값만 가지므로데이터 오류라 볼 수 있다.
- 3개 밖에 없어서 삭제를 고려했으나, 위에서 결측치 처리한 것처럼 knn알고리즘으로 채워본 결과, 0.8 으로 채워져 1로 반올림 하여 채워주었다.

	Original Inflight Service	Replacement Inflight Service
466	0	0.8
51397	0	0.8
88714	0	0.8

```
train_df['inflight_service'] = train_df['inflight_service'].replace(0.8, 1)
```

- ‘departure_delay’, ‘arrival_delay’ 같은 경우 z-score 3 이상인 것을 기준으로 이상치 탐지 했을 때 약 2천개가 이상치로 분류되었다.



- boxplot으로 시각화하여 데이터 분포 확인
- “*_delay” 열은 이상치를 제거하기 보단 범주화 시도
- 범주를 1, 2, 3으로 나누어 본 결과, 0-180분의 데이터가 압도적(90%)으로 많았기에 0-180분의 데이터를 세분화하였다.

Training Dataset:

- **Departure Delay:**
  - 1.1 (≤ 60 minutes): 96,665 occurrences
  - 1.2 (61-120 minutes): 4,699 occurrences
  - 1.3 (121-180 minutes): 1,485 occurrences
  - 2.0 (181-360 minutes): 936 occurrences
  - 3.0 (> 360 minutes): 119 occurrences

- **Arrival Delay:**
  - 1.1 (≤ 60 minutes): 96,515 occurrences
  - 1.2 (61-120 minutes): 4,744 occurrences
  - 1.3 (121-180 minutes): 1,553 occurrences
  - 2.0 (181-360 minutes): 963 occurrences
  - 3.0 (> 360 minutes): 129 occurrences

Testing Dataset:

- **Departure Delay:**
  - 1.1 (≤ 60 minutes): 24,240 occurrences
  - 1.2 (61-120 minutes): 1,183 occurrences
  - 1.3 (121-180 minutes): 318 occurrences
  - 2.0 (181-360 minutes): 204 occurrences
  - 3.0 (> 360 minutes): 31 occurrences

- **Arrival Delay:**
  - 1.1 (≤ 60 minutes): 24,197 occurrences
  - 1.2 (61-120 minutes): 1,197 occurrences
  - 1.3 (121-180 minutes): 343 occurrences
  - 2.0 (181-360 minutes): 208 occurrences
  - 3.0 (> 360 minutes): 31 occurrences

7. 모델링

- **그래디언트 부스팅 알고리즘 (Gradient Boosting XGBoost)**

```
# XGBoost 분류기 초기화
# GridSearch를 수행하여 최적을 파라미터를 찾아 적용
xgb_model = XGBClassifier(
    learning_rate= 0.1,
    max_depth=7,
    n_estimators=160,
    min_child_weight = 3,
    gamma=0.1,
    reg_alpha= 0,
    reg_lambda=0.1,
    subsample= 1.0,
    colsample_bytree= 0.8,
    scale_pos_weight= 1
)

# 훈련 데이터를 사용하여 모델 훈련 및 검증 세트에서 평가
xgb_model.fit(X_train, y_train)

# 검증 세트에서 예측 수행
y_test_pred = xgb_model.predict(X_test)

# 모델 평가
accuracy = accuracy_score(y_test, y_test_pred)
print(f'accuracy: {accuracy:.5f}')

# 분류 보고서 출력
print("classification report:\n", classification_report(y_test, y_test_pred))
```

```
>>> accuracy: 0.96497
>>> classification report:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97     14573
           1       0.97      0.94      0.96     11403

    accuracy                           0.96     25976
   macro avg       0.97      0.96      0.96     25976
weighted avg       0.97      0.96      0.96     25976
```

초기 모델을 학습 시킨 결과: accuracy = 0.96478

최적의 하이퍼 파라미터 적용한 모델링 결과: acurracy = 0.96497

GridSearch를 수행함으로써 모델의 정확도를 0.00019 상승

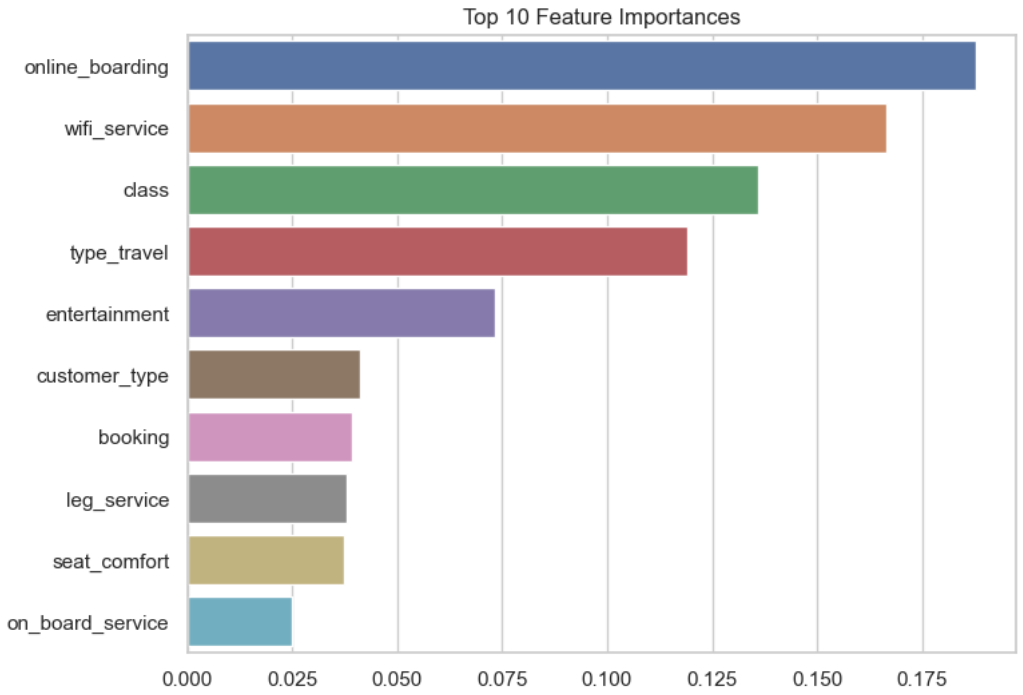
- **랜덤 포레스트 (Random Forest)**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, make_scorer

# Random Forest 초기화
rf1 = RandomForestClassifier(n_estimators =100, max_depth=12,
                             min_samples_leaf = 8, min_samples_split = 8,
                             random_state =11, n_jobs = -1)

# 모델 훈련 및 검증 세트 평가
rf1.fit(X_train, y_train)

# 모델 평가
y_pred =rf1.predict(X_test)

# Accuracy
print("Accuracy score of model : {0:4f}".format(accuracy_score(y_test,y_pred)))

# f1-score 출력
print("F1 score of model : {0:4f}".format(f1_score(y_test,y_pred, average='macro')))
```

```
Accuracy score of model : 0.952726
F1 score of model : 0.951923
```

- 상위 10개 피처 시각화 후 해당 피처만 적용하여 accuracy 확인

```
import matplotlib.pyplot as plt
import seaborn as sns

# Top 10 Feature Importance 시각화
ftr_importances_values = rf1.feature_importances_
ftr_importances = pd.Series(ftr_importances_values, index=train_df.drop('satisfaction', axis=1).columns)
ftr_top10 = ftr_importances.sort_values(ascending=False)[:10]

plt.figure(figsize=(8,6))
plt.title('Top 10 Feature Importances')
sns.barplot(x=ftr_top10, y=ftr_top10.index)
plt.show()
```




```
#상위 10개 중요도를 가진 피처 선택
top_features = ftr_importances.sort_values(ascending=False).head(10).index.tolist()

#선택된 피처로 데이터셋을 구성
X_train_selected = train_df[top_features]
X_test_selected = test_df[top_features]

# 새로운 Random Forest 모델 학습
rf_selected = RandomForestClassifier(n_estimators=100, max_depth=12, min_samples_leaf=8, min_samples_split=8, random_state=11, n_jobs=
rf_selected.fit(X_train_selected, y_train)

# 새로운 모델 평가
pred_selected = rf_selected.predict(X_test_selected)
print("Accuracy score of model with selected features: {0:.4f}".format(accuracy_score(y_test, pred_selected)))
print("F1 score of model with selected features: {0:.4f}".format(f1_score(y_test, pred_selected, average='macro')))
```

```
Accuracy score of model with selected features: 0.9456
F1 score of model with selected features: 0.9447
# 10개만 선택하였을 때 오히려 줄어드는 것을 확인함
```

- **서포트 벡터 머신 (Support Vector Machines, SVM)**

하이퍼 파라미터 튜닝을 시도하였으나, 커널이 죽는 문제 발생 -> 해결 못함

정확도 또한 0.8978 으로 다른 모델에 비해 떨어져 채택하지 않았다.

- **Cat Boosting**
  - optuna를 이용한 하이퍼 파라미터 튜닝
  - 기존 Grid search를 이용한 하이퍼 파라미터 튜닝보다 수행 시간이 짧다는 장점이 있다.
  - 해당 방법으로 구한 하이퍼 파라미터들을 모델에 대입

```
from sklearn.metrics import accuracy_score, classification_report

accuracy =[]
model_names =[]


X= train_x
y= train_y
categorical_features_indices = np.where(X.dtypes != np.float)[0]


model = CatBoostClassifier(
    verbose=False,
    learning_rate=0.23540275740381159,
    objective= 'CrossEntropy',
    colsample_bylevel= 0.09478805616996021,
    depth= 7,
    boosting_type= 'Ordered',
    bootstrap_type= 'MVS',
)

model.fit(train_x, train_y,cat_features=categorical_features_indices,eval_set=(test_x, test_y))

y_pred = model.predict(test_x)
accuracy.append(round(accuracy_score(test_y, y_pred),4))
print(classification_report(test_y, y_pred))

model_names = ['Catboost_tuned']
result_df6 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df6

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     14573
           1       0.97      0.95      0.96     11403

    accuracy                           0.97     25976
   macro avg       0.97      0.96      0.96     25976
weighted avg       0.97      0.97      0.97     25976

Accuracy
Catboost_tuned  0.9653
```

하이퍼 파라미터 튜닝 전 정확도: 0.9649

하이퍼 파라미터 튜닝 후 정확도: 0.9653

정확도가 소폭 상승하였다.

- **KNN (K-최근접 이웃 알고리즘)**
  - 기본적으로 k=5

  5NN Score: 0.9170
  F1 score of model: 0.9150

  - GridSearch로 하이퍼 파라미터 튜닝

  GridSearchCV hightest mean accuracy value : 0.9153
  GridSearchCV best hyperparameter : {'algorithm': 'brute', 'n_neighbors': 7}

  7NN Score: 0.9168
  F1 score of model: 0.9149

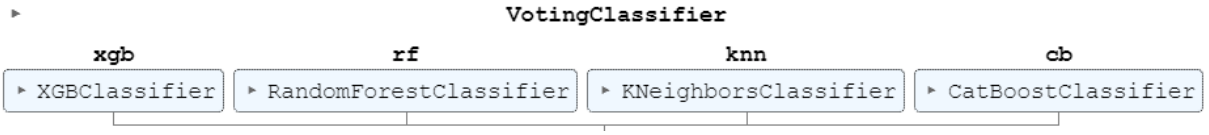
  - 하지만 Grid Search를 통해 구한 파라미터인 k=7보다 k=5일 때 accuracy가 더 높았다.
  - 차원축소 PCA를 이용해봤지만, 오히려 차원을 축소하였을 때 accuracy는 낮아졌다.
  - 정확도를 올려보기 위해 다른 파라미터를 조정해 본 결과 KNN의 파라미터 중 metric, weights를 변경해주었더니 정확도가 올랐다.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, f1_score

# KNN 모델 생성 및 훈련
knn = KNeighborsClassifier(algorithm='brute', n_neighbors=7, metric='manhattan', weights='distance')
knn.fit(X_train, y_train)

# 테스트 데이터에 대한 예측
knn_pred = knn.predict(X_test)

# 정확도 및 F1 점수 출력
print("{0}NN Score: {1:.4f}".format(5, accuracy_score(y_test, knn_pred)))
print("F1 score of model: {0:.4f}".format(f1_score(y_test, knn_pred, average='macro')))
```

  7NN Score: 0.9292
  F1 score of model: 0.9278

- **앙상블 모델 (Ensemble Models)**

Soft Voting Classification




```
# 최종 모델링 및 평가
from sklearn.metrics import classification_report, accuracy_score, f1_score

y_pred=voting_clf.predict(X_test)
print("Voting Accuracy Score :{0:.5f}".format(accuracy_score(y_test,y_pred)))
print("F1 score of model : {0:.5f}".format(f1_score(y_test,y_pred,average='macro')))
```

```
>>> Voting Accuracy Score :0.96304
>>> F1 score of model : 0.96235
```

결론

분류 프로젝트을 진행하며 처음부터 높은 정확도를 얻었음에도 불구하고, 차원 축소와 변수 선택 등을 통한 추가적인 시도는 오히려 성능을 저하시켰다. 이는 데이터셋이 본래 가지고 있는 특성들이 모두 중요한 역할을 했을 가능성을 나타낸다고 생각한다. 즉, 중요한 정보가 손실되었을 수 있고, 변수 간의 복잡한 상호작용이 제대로 반영되지 않았을 수 있다.

이번 프로젝트롤 통해 데이터의 기본 구조를 잘 이해하고, 각 변수의 중요성을 신중하게 고려해야 한다는 것을 깨닫게 해주었다. 또한, 모든 분석에서 최적의 결과를 얻기 위해서는 데이터의 특성과 모델의 성격을 잘 파악하여 모델의 파라미터를 조정하는 것이 예측 성능을 최적화하는 것이 핵심적인 역할을 하는 것을 알 수 있었다.