고객 행동 예측
(구매)

컴퓨터공학전공
2018108258
문준혁

목차

개요

데이터 소개

	User ID	Gender	Age	EstimatedSalary	Purchased
1	15624510	Male	19	19000	0
2	15810944	Male	35	20000	0
3	15668575	Female	26	43000	0
4	15603246	Female	27	57000	0
5	15804002	Male	19	76000	0
6	15728773	Male	27	58000	0
7	15598044	Female	27	84000	0
8	15694829	Female	32	150000	1
9	15600575	Male	25	33000	0
10	15727311	Female	35	65000	0
11	15570769	Female	26	80000	0
12	15606274	Female	26	52000	0
13	15746139	Male	20	86000	0
14	15704987	Male	32	18000	0
15	15628972	Male	18	82000	0
16	15697686	Male	29	80000	0
17	15733883	Male	47	25000	1
18	15617482	Male	45	26000	1
19	15704583	Male	46	28000	1
20	15621083	Female	48	29000	1
21	15649487	Male	45	22000	1
22	15736760	Female	47	49000	1
23	15714658	Male	48	41000	1
24	15599081	Female	45	22000	1
25	15705113	Male	46	23000	1
26	15631159	Male	47	20000	1
27	15792818	Male	49	28000	1

Features (고객 400명)

- User ID
- Gender
- Age
- EstimatedSalary

Target

- Purchased

데이터 전처리

라이브러리

```
# import requirement libraries
# 라이브러리 불러오기

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
import itertools

# optional

# 경고 메시지 무시하기
import warnings
warnings.filterwarnings('ignore')

# 스타일 지정(그래프, 차트)
plt.style.use('_mpl-gallery')
```

```
# 머신러닝_사이킷런
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

데이터 전처리

라이브러리

pandas

데이터 분석 라이브러리
2차원(데이터프레임) 데이터를
효율적으로 가공/처리

numpy

배열 라이브러리
고차원의 배열을 손쉽게 만들고
조작

matplotlib

과학계산용 그래프를 그리는
라이브러리

seaborn

matplotlib 기반의 시각화
라이브러리

itertools

파이썬 내장 라이브러리
효율적인 루핑을 위한
이터레이터를 만드는 함수

sklearn

머신러닝 라이브러리

데이터 전처리

데이터 불러오기

```
# import dataset
# 데이터 세트 가져오기

data = pd.read_csv('./Customer_Behaviour.csv')

print(f"shape: {data.shape}") # 데이터의 (행, 열) 출력

data.head() # 데이터의 상위 5개의 행 출력
```

shape: (400, 5)

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

5 rows × 5 columns 새 탭에서 열기

```
# 2차원 데이터 구조

df = pd.DataFrame(data)
df
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1

400 rows × 5 columns 새 탭에서 열기

데이터 전처리

데이터 정보

```
# 데이터에 대한 정보 출력

df.info()
```

```
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

정보

- 누락된 값 없음
- 수치형 데이터 : 고객 ID, 나이, 급여, 구매 여부
- 범주형 데이터 : 성별

데이터 전처리

결측치 처리

```
# 데이터에 있는 값 중에 null이 있는지 체크

df.isnull().sum().to_frame('NaN value').T

# df.isnull().sum() => df의 null 값의 개수
# to_frame => df.isnull().sum()을 데이터 프레임으로 반환
# T => 행과 열 바꾸기
```

	User ID	Gender	Age	EstimatedSalary	Purchased
NaN value	0	0	0	0	0

```
# check count of unique values in each columns
# 각 특징의 유니크한 값(중복 제외)의 개수 체크

for col in df:
    print(f"{col}: {df[col].nunique()}")

# nunique() => 데이터의 고유 값들의 수 체크
```

```
User ID: 400
Gender: 2
Age: 43
EstimatedSalary: 117
Purchased: 2
```

데이터 전처리

결측치 처리

```
# more details
# 값이 number인 데이터를 요약(개수, 평균, 표준편차, 최소, 최대)

df.describe(include=[np.number]).T
```

	count	mean	std	min	25%	50%	75%	max
User ID	400.0	1.569154e+07	71658.321581	15566689.0	15626763.75	15694341.5	15750363.0	15815236.0
Age	400.0	3.765500e+01	10.482877	18.0	29.75	37.0	46.0	60.0
EstimatedSalary	400.0	6.974250e+04	34096.960282	15000.0	43000.00	70000.0	88000.0	150000.0
Purchased	400.0	3.575000e-01	0.479864	0.0	0.00	0.0	1.0	1.0

```
# 값이 object인 데이터를 요약(개수, 유니크한 값,데이터에서 가장 많이 나온 값, 가장 많이 나온 값의 개수)

df.describe(include=[object]).T
```

	count	unique	top	freq
Gender	400	2	Female	204

정보

- 나이, 급여의 범위
- 성별 : 남성과 여성의 수가 거의 같음(196, 204)
- 구매 여부 : 2개의 값(0, 1)
- 표준화가 필요함(데이터 변수들의 값 범위가 다름)

데이터 전처리

결측치 처리

```
# Drop User ID columns
# 사용자 ID는 예측에 필요 없으므로 삭제

df.drop('User ID', axis=1, inplace=True)
df
```

	Gender	Age	EstimatedSalary	Purchased
6	Female	27	84000	0
7	Female	32	150000	1
8	Male	25	33000	0
9	Female	35	65000	0
10	Female	26	80000	0
11	Female	26	52000	0
12	Male	20	86000	0
13	Male	32	18000	0

400 rows × 4 columns 새 탭에서 열기

고객 ID는 필요가 없으므로 삭제

```
# convert categoriacl feature to numerical:
 # only Gender is categorical
# 성별만 값이 number가 아니므로 전환((male,female) → (0,1))

df['Gender'] = df['Gender'].replace(['Male', 'Female'], [0, 1])
df
```

	Gender	Age	EstimatedSalary	Purchased
0	0	19	19000	0
1	0	35	20000	0
2	1	26	43000	0
3	1	27	57000	0
4	0	19	76000	0
5	0	27	58000	0
6	1	27	84000	0
7	1	32	150000	1

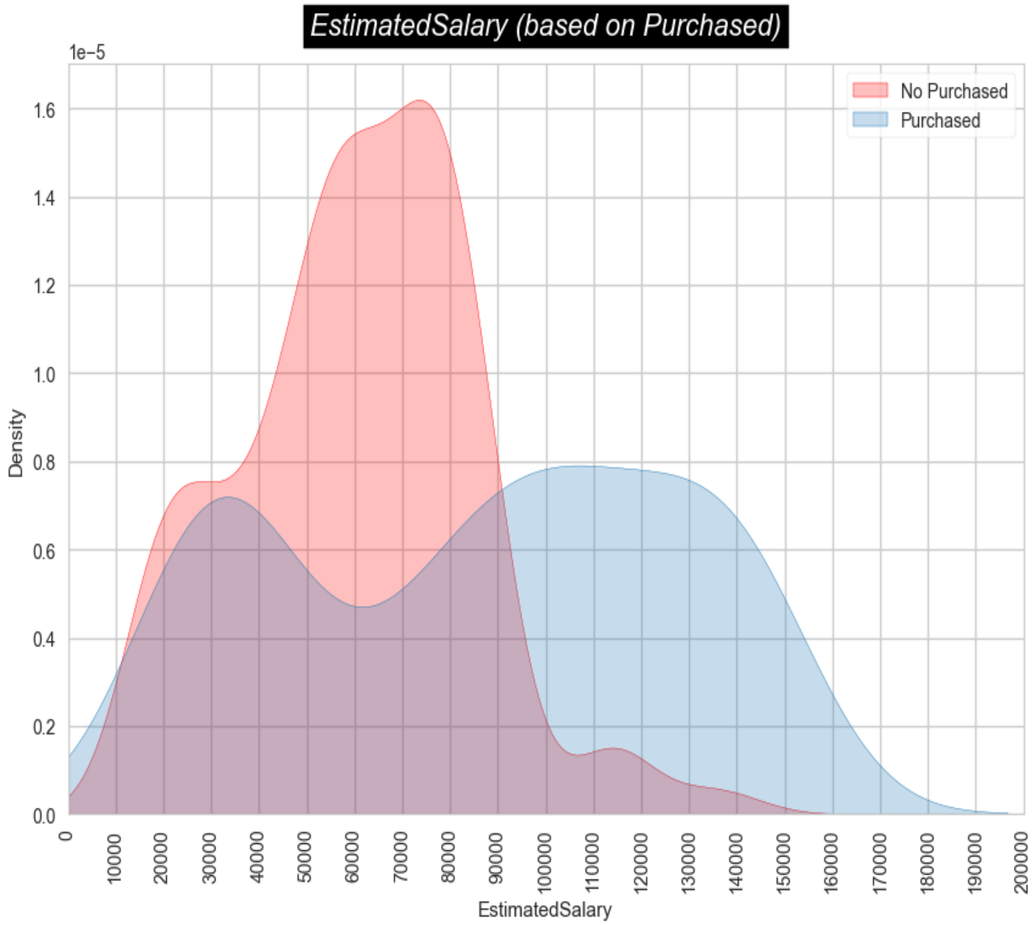
400 rows × 4 columns 새 탭에서 열기

범주형 데이터 → 수치형 데이터

EDA

급여(구매한 기준)

```
# check distribution of EstimatedSalary (based on Purchased)
# 추정 급여에 따른 분포 체크(구매한 기준)

font = {'fontsize':16, 'fontstyle':'italic', 'backgroundcolor':'black', 'color':'orange'}
%matplotlib inline
plt.style.use('seaborn-notebook')

sns.kdeplot(df.loc[df['Purchased'] == 0, 'EstimatedSalary'], label='No Purchased', shade=True, color='red')
sns.kdeplot(df.loc[df['Purchased'] == 1, 'EstimatedSalary'], label='Purchased', shade=True)
plt.title('KDE of EstimatedSalary (based on Purchased)', fontdict=font, pad=15)
plt.xticks(np.arange(0,200001,10000), rotation=90)
plt.xlim([0,200001])
plt.legend()
plt.show()


# %matplotlib inline => 주피터 노트북을 실행한 브라우저에서 바로 그림을 볼수 있게 해주는 코드
# sns.kdeplot() => 히스토그램보다 부드러운 형태의 분포 곡선(커널 밀도 추정을 사용하여 일변량 또는 이변량 분포를 플로팅)
# plt.xticks(), plt.xlim() => x축 눈금(간격 구분), x축 범위 지정
# plt.legend(), plt.show() => 범례, 그래프
```



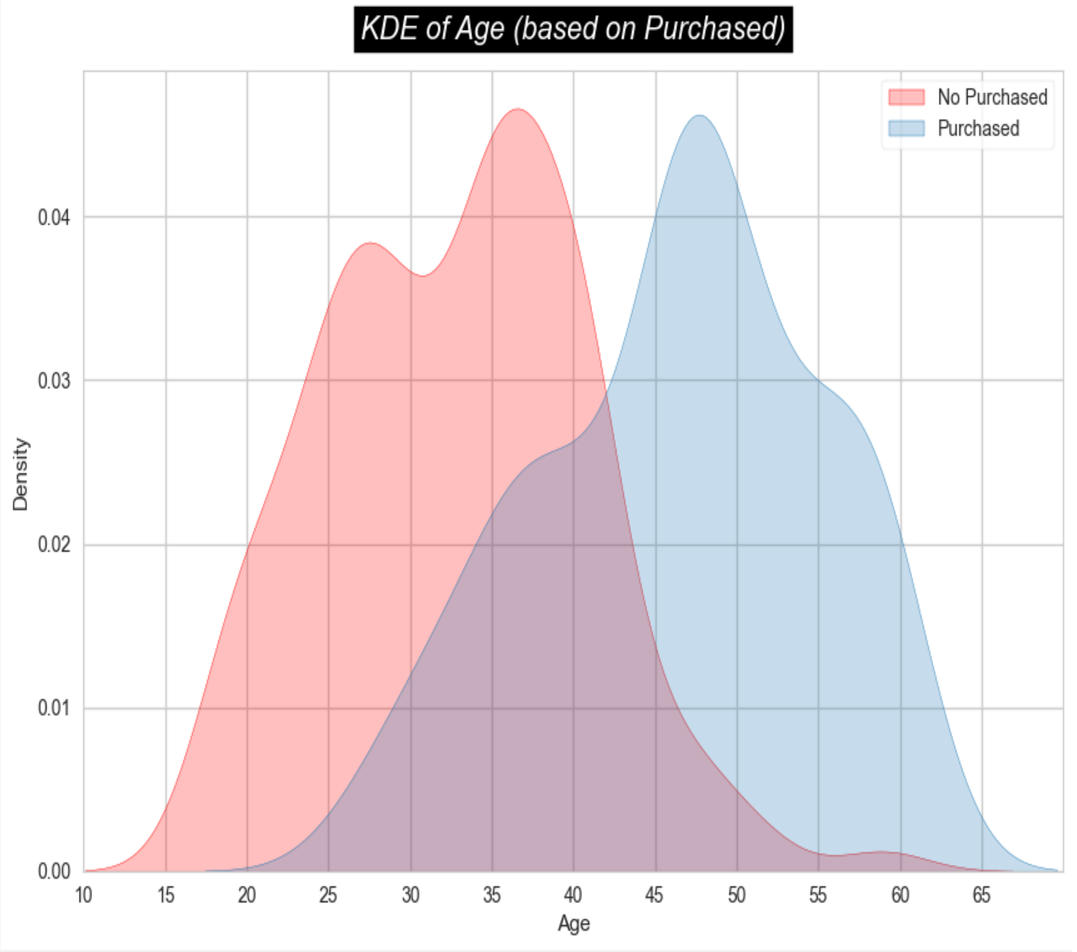
- 급여가 높을수록 구매
- 급여가 40000~90000인 사람들은 구매가 적음

EDA

나이(구매한 기준)

```
# check distribution of Age (based on Purchased)
# 나이에 따른 분포 체크(구매한 기준)


%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Purchased'] == 0, 'Age'], label='No Purchased', shade=True, color='red')
sns.kdeplot(df.loc[df['Purchased'] == 1, 'Age'], label='Purchased', shade=True)
plt.title('KDE of Age (based on Purchased)', fontdict=font, pad=15)
plt.xticks(np.arange(0,70,5))
plt.xlim([10,70])
plt.legend()
plt.show()
```

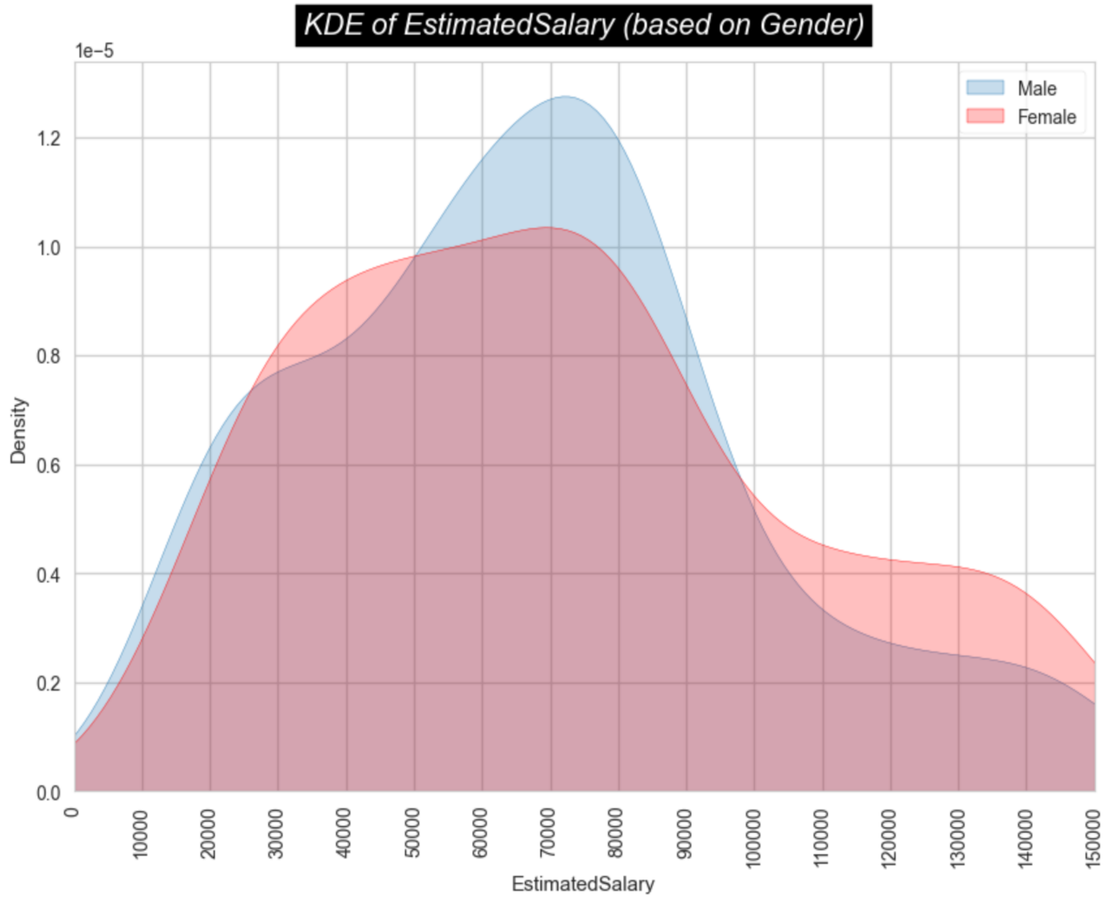
- 구매를 한 사람들의 나이가 구매를 하지 않은 사람들보다 나이가 많음
- 43세 이상부터 구매를 하는 사람이 구매를 하지 않는 사람보다 많아짐



EDA

급여(성별 기준)

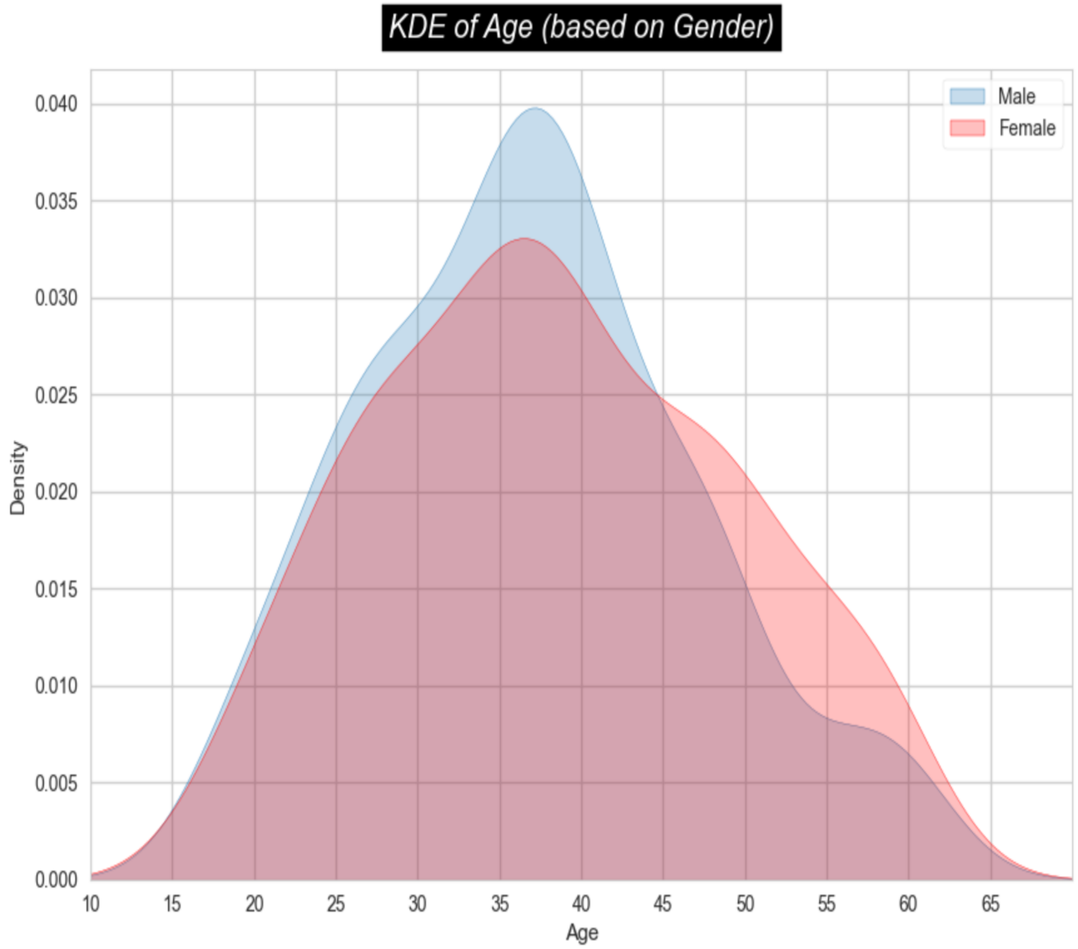
```
# check distribution of EstimatedSalary (based on Gender)
# 추정 급여에 따른 분포 체크(성별 기준)

%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Gender'] == 0, 'EstimatedSalary'], label='Male', shade=True)
sns.kdeplot(df.loc[df['Gender'] == 1, 'EstimatedSalary'], label='Female', shade=True, color='red')
plt.title('KDE of EstimatedSalary (based on Gender)', fontdict=font, pad=15)
plt.xticks(np.arange(0,150001,10000), rotation=90)
plt.xlim([0,150001])
plt.legend()
plt.show()
```



EDA

나이(성별 기준)

```
# check distribution of Age (based on Gender)
# 나이에 따른 분포 체크(성별 기준)

%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Gender'] == 0, 'Age'], label='Male', shade=True)
sns.kdeplot(df.loc[df['Gender'] == 1, 'Age'], label='Female', shade=True, color='red')
plt.title('KDE of Age (based on Gender)', fontdict=font, pad=15)
plt.xticks(np.arange(0,70,5))
plt.xlim([10,70])
plt.legend()
plt.show()
```

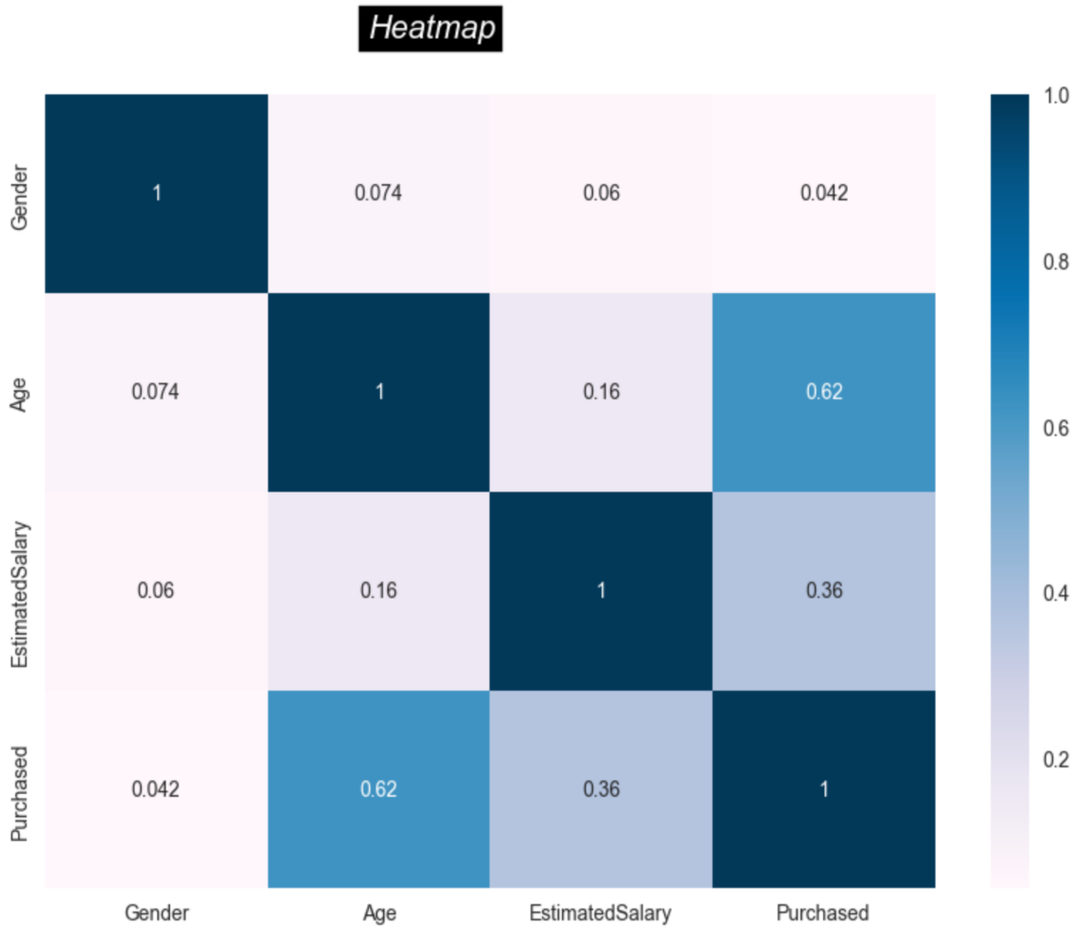


EDA

히트맵

```
# heatmap
sns.heatmap(df.corr(), cmap='PuBu', annot=True)
plt.suptitle('Heatmap', y=1.09, x=0.35, **font)
plt.show()
```

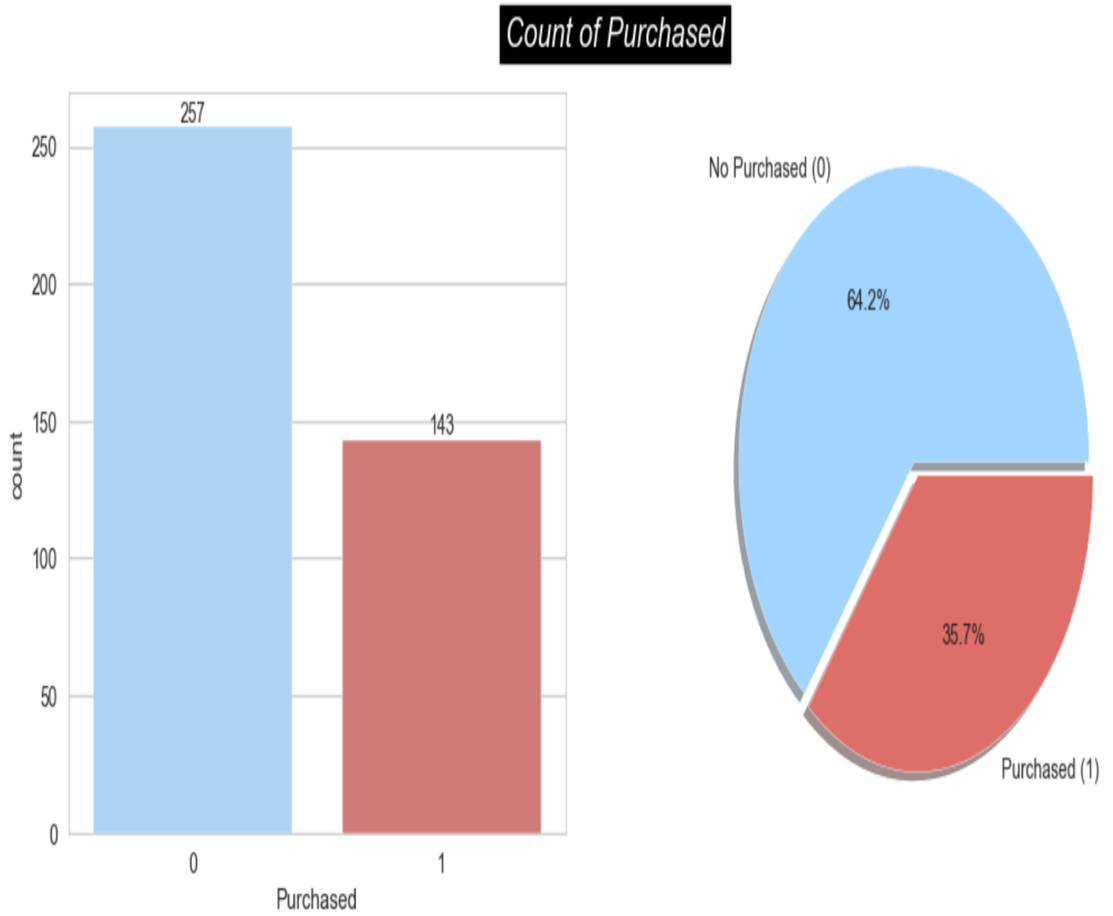
- 변수간 상관관계
- 구매여부와 나이가 상관관계가 가장 높음(0.62)
- 구매여부와 급여가 두번째로 높음(0.36)



EDA

일변량 분석(구매 여부)

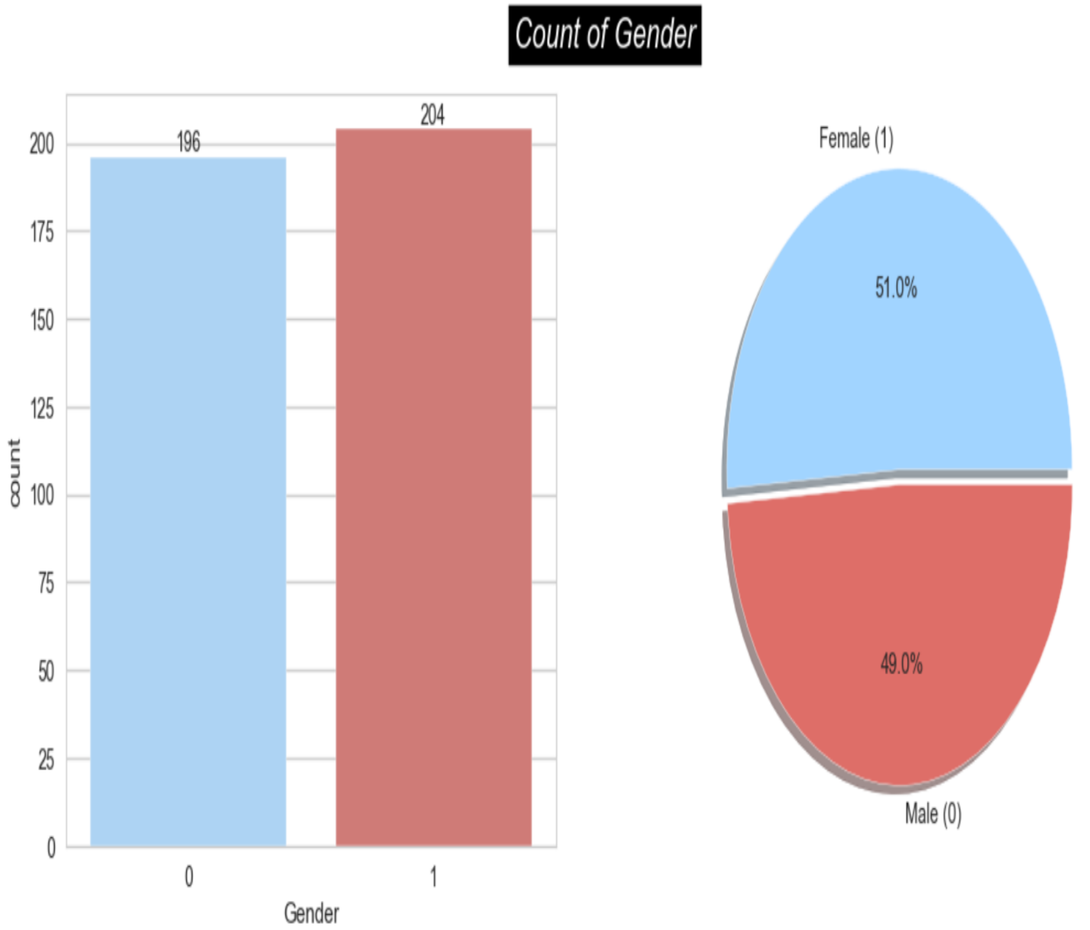
```
# count based on Purchased (countplot)
fig, axes = plt.subplots(1,2,figsize=(10,4))

sns.countplot(data=df, x='Purchased', ax=axes[0], palette=['#A1D4FF','#DE6E68'])
for container in axes[0].containers:
    axes[0].bar_label(container)

# count based on Purchased (pie chart)
slices = df.Purchased.value_counts().values
activities = ['No Purchased (0)', 'Purchased (1)']
axes[1].pie(slices, labels=activities, colors=['#A1D4FF','#DE6E68'], shadow=True, explode=[0,0.05], autopct='%1.1f%%')

plt.suptitle('Count of Purchased', y=1.09, **font)
plt.show()

# 구매를 결정한 사람과 구매를 결정하지 않은 사람
```



EDA

일변량 분석(성별)

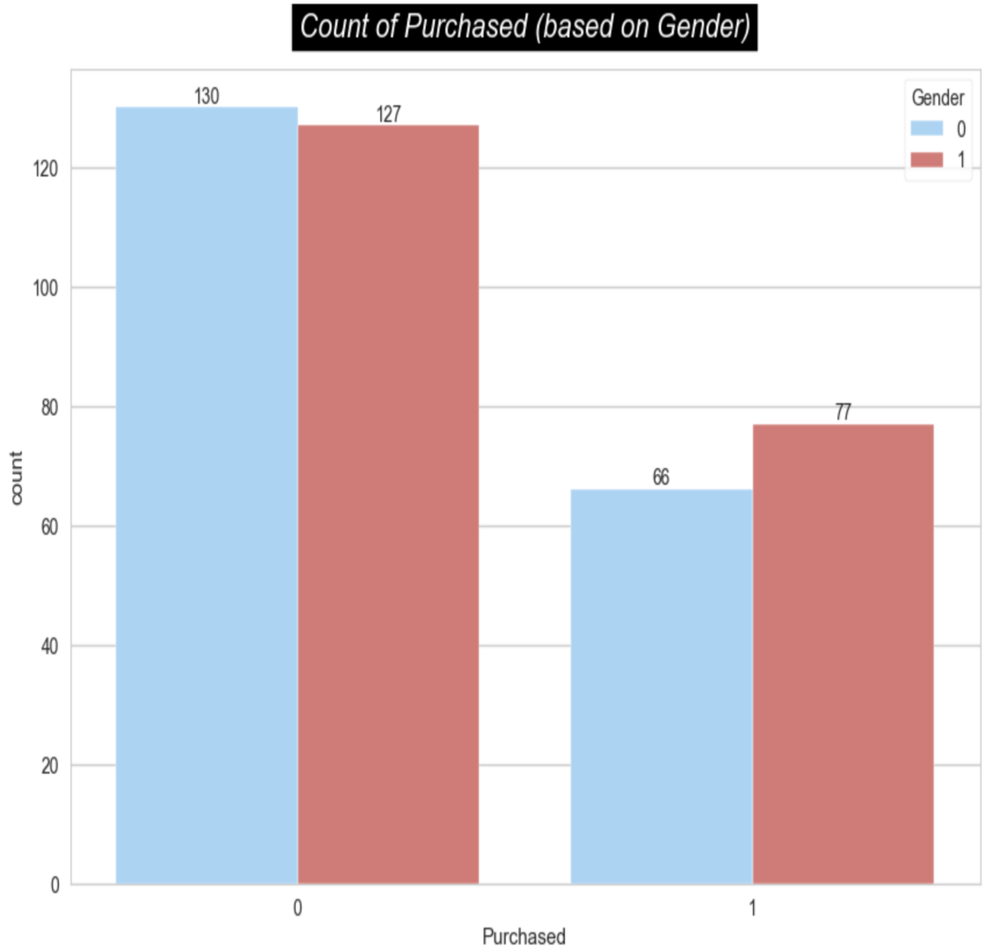
```
# count based on Gender (countplot)
fig, axes = plt.subplots(1,2,figsize=(10,4))

sns.countplot(data=df, x='Gender', ax=axes[0], palette=['#A1D4FF','#DE6E68'])
for container in axes[0].containers:
    axes[0].bar_label(container)

# count based on Gender (pie chart)
slices = df.Gender.value_counts().values
activities = ['Female (1)', 'Male (0)']
axes[1].pie(slices, labels=activities, colors=['#A1D4FF','#DE6E68'], shadow=True, explode=[0,0.05], autopct='%1.1f%%')

plt.suptitle('Count of Gender', y=1.09, **font)
plt.show()

# 성별
```



EDA

이변량 분석(성별 + 구매)

```
# count of purchased based on Gender
%matplotlib inline
ax = sns.countplot(data=df, x='Purchased', hue='Gender', palette=['#A1D4FF','#DE6E68'])
for container in ax.containers:
    ax.bar_label(container)
plt.title('Count of Purchased (based on Gender)', fontdict=font, pad=15)
plt.show()

# 구매 여부 + 성별
```

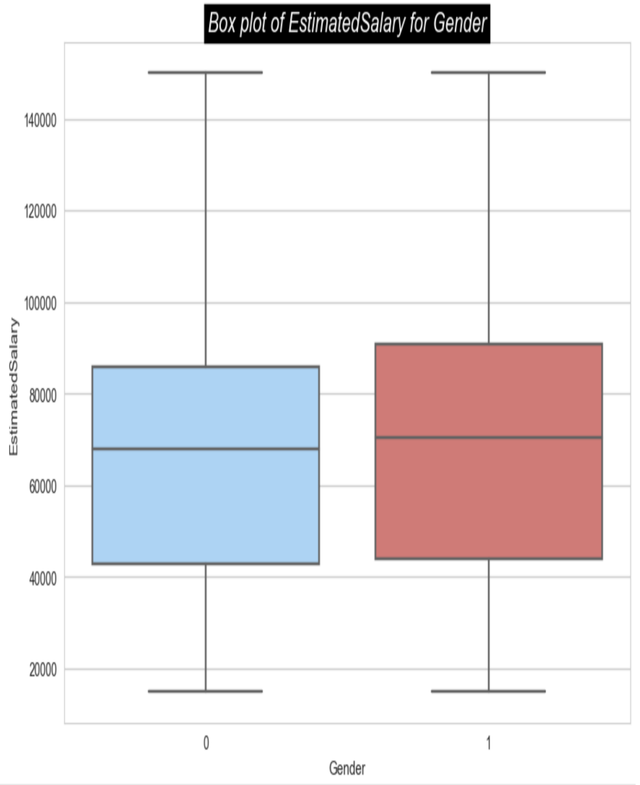


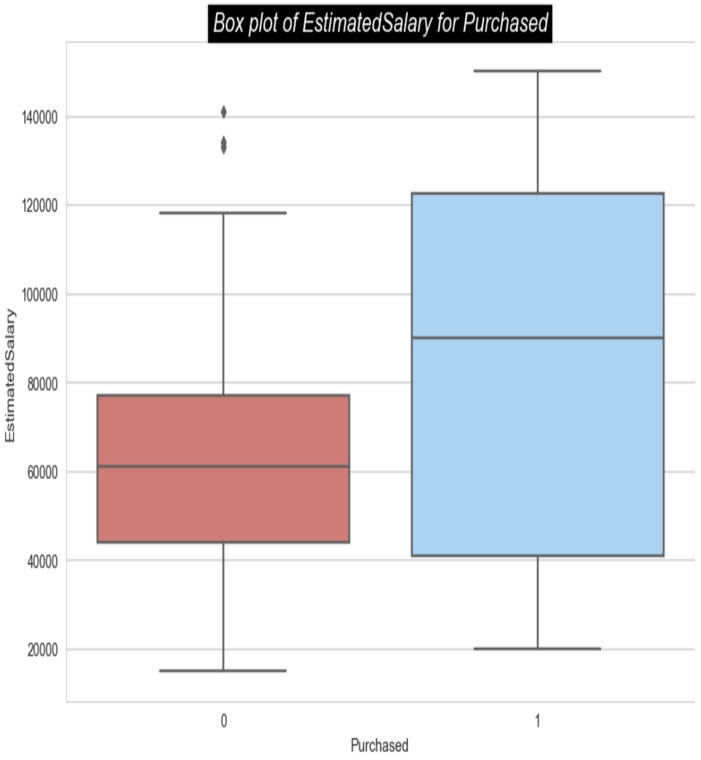
- 제품을 구매한 사람들 중에는 남성보다 여성이 많음
- 제품을 구매하지 않은 사람들 중에는 남성이 더 많음

EDA

이변량 분석(성별, 구매 => 급여)

```
# draw box plot of Estimated salary for male(0) or female(1) Gender
sns.boxplot(data=df, x='Gender', y='EstimatedSalary', palette=['#A1D4FF', '#DE6E68'])
plt.title(f'Box plot of EstimatedSalary for Gender', fontdict=font)
plt.show()

# draw box plot of Estimated salary for no purchased(0) or purchased(1)
sns.boxplot(data=df, x='Purchased', y='EstimatedSalary', palette=['#DE6E68', '#A1D4FF'])
plt.title(f'Box plot of EstimatedSalary for Purchased', fontdict=font)
plt.show()

# 성별, 구매 여부 => 박스 플롯(추정 급여)
# box plot => 최대값, 최소값, 중위수, 제1 사분위(25%), 제3 사분위(75%), 제2 사분위(50%)
```

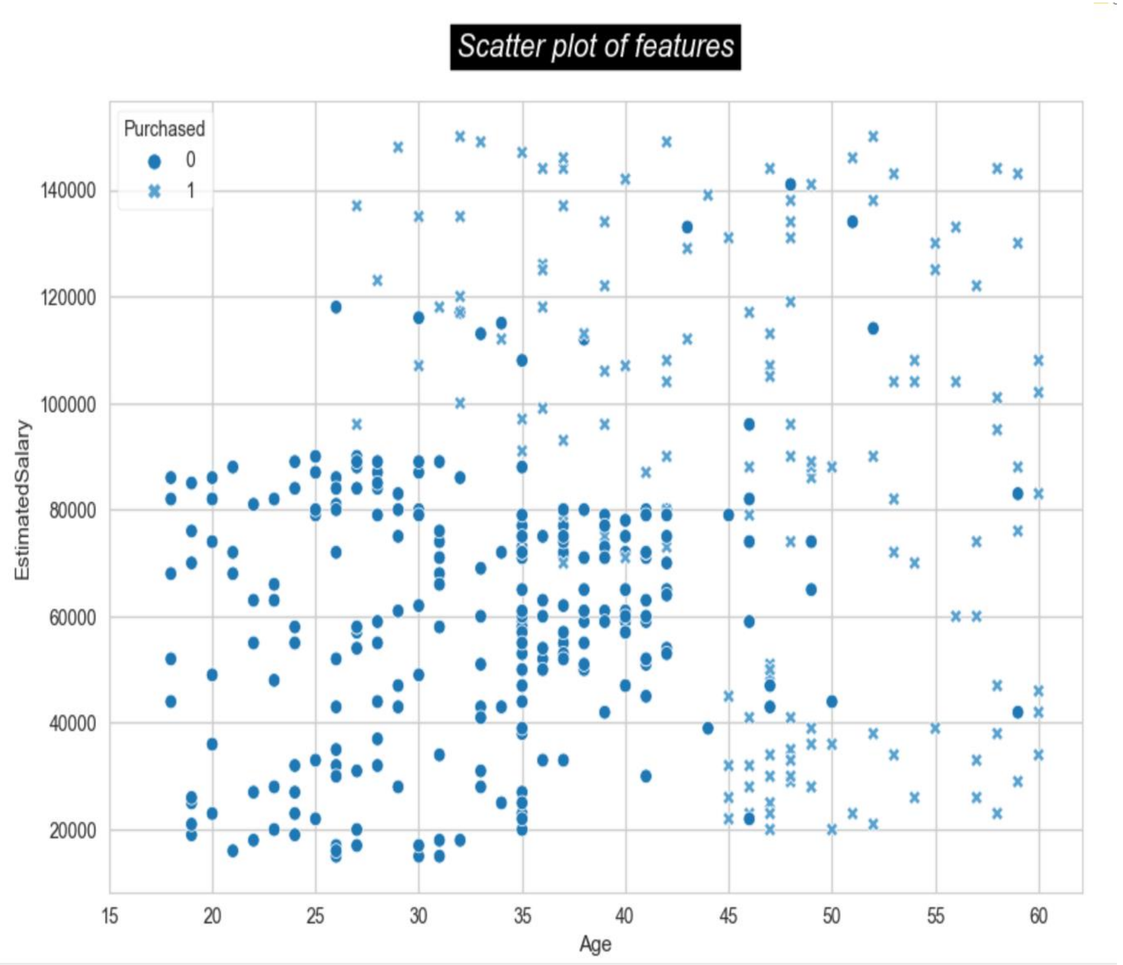




EDA

다변량 분석

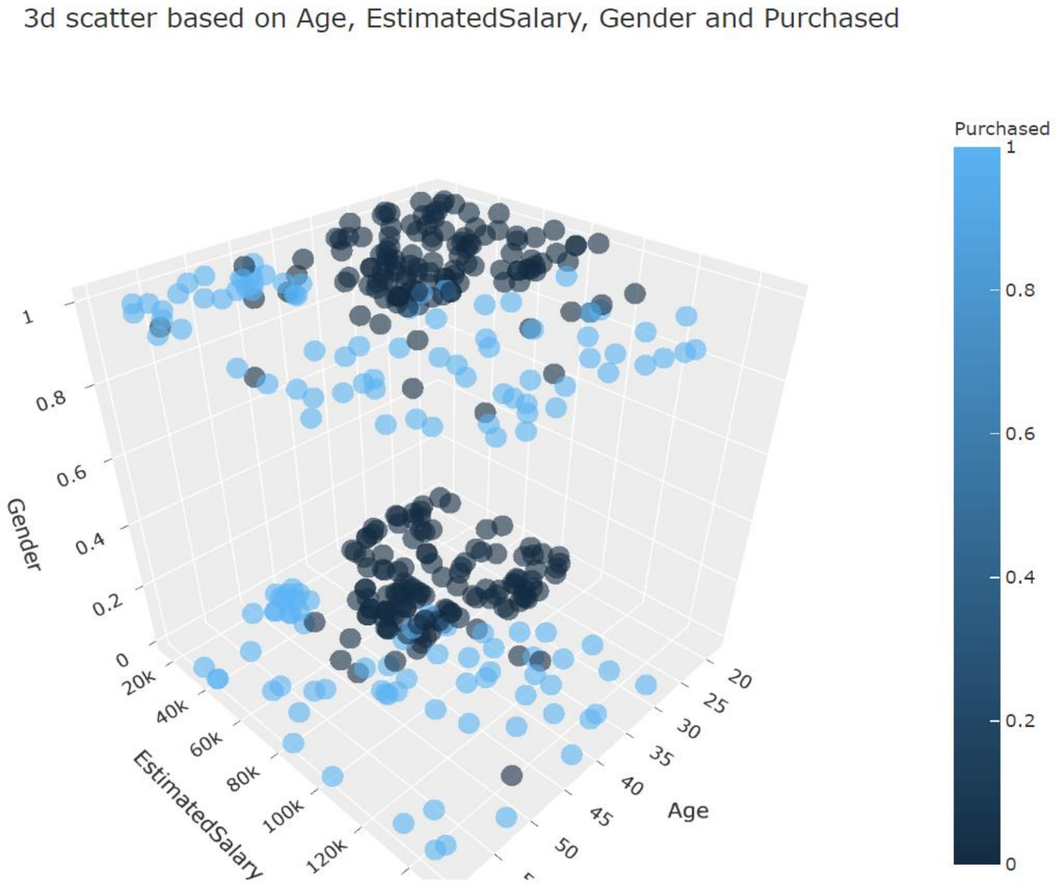
```
%matplotlib inline

# check feature correlation


sns.scatterplot(data=df,x='Age', y='EstimatedSalary', hue='Purchased', style='Purchased')

plt.title('Scatter plot of features', y=1.04, fontdict=font)

plt.xticks(np.arange(15,65,5))

plt.show()


# 나이 + 급여 => 구매 여부
```



EDA

다변량 분석(3D)

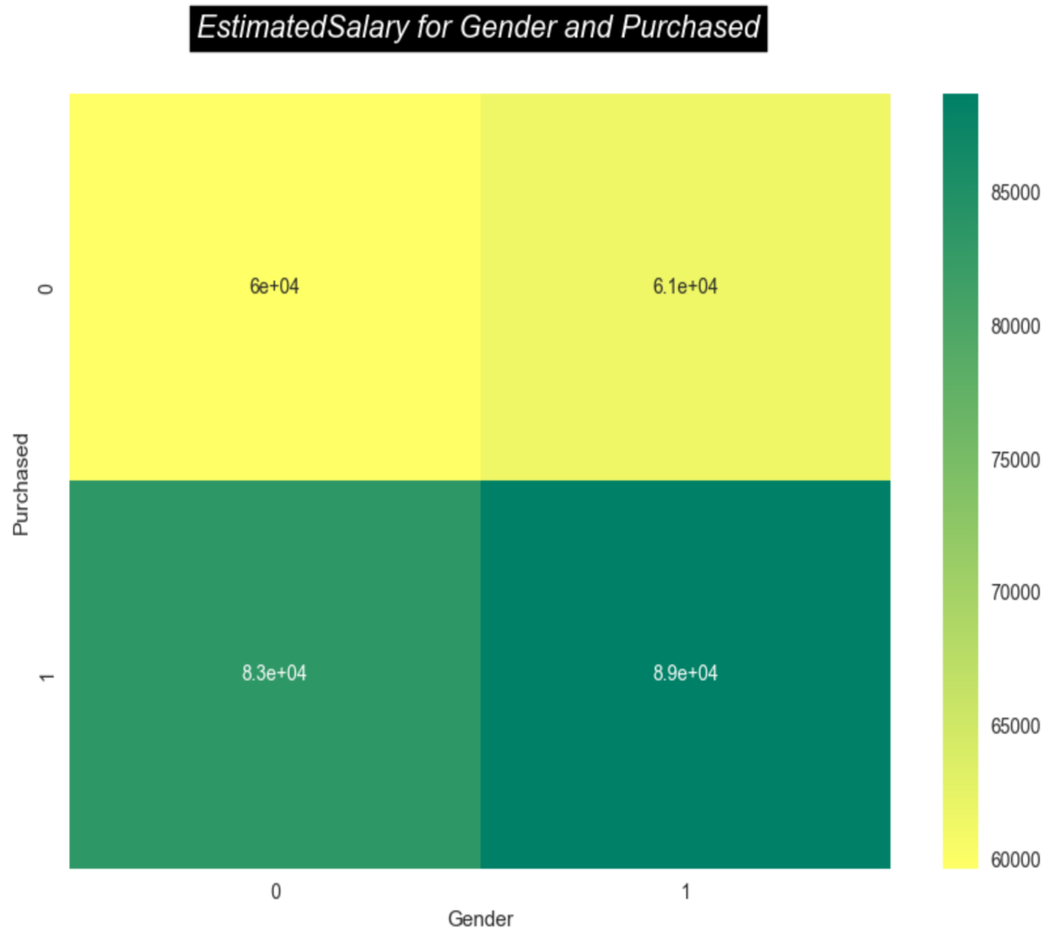
```
fig = px.scatter_3d(
        data_frame=df,
        x='Age',
        y='EstimatedSalary',
        z='Gender',
        color='Purchased',
        template='ggplot2',
        opacity=0.6,
        height=700,
        title=f'3d scatter based on Age, EstimatedSalary, Gender and Purchased'
)

pio.show(fig)
```



EDA

다변량 분석(성별 + 구매 => 급여 평균)

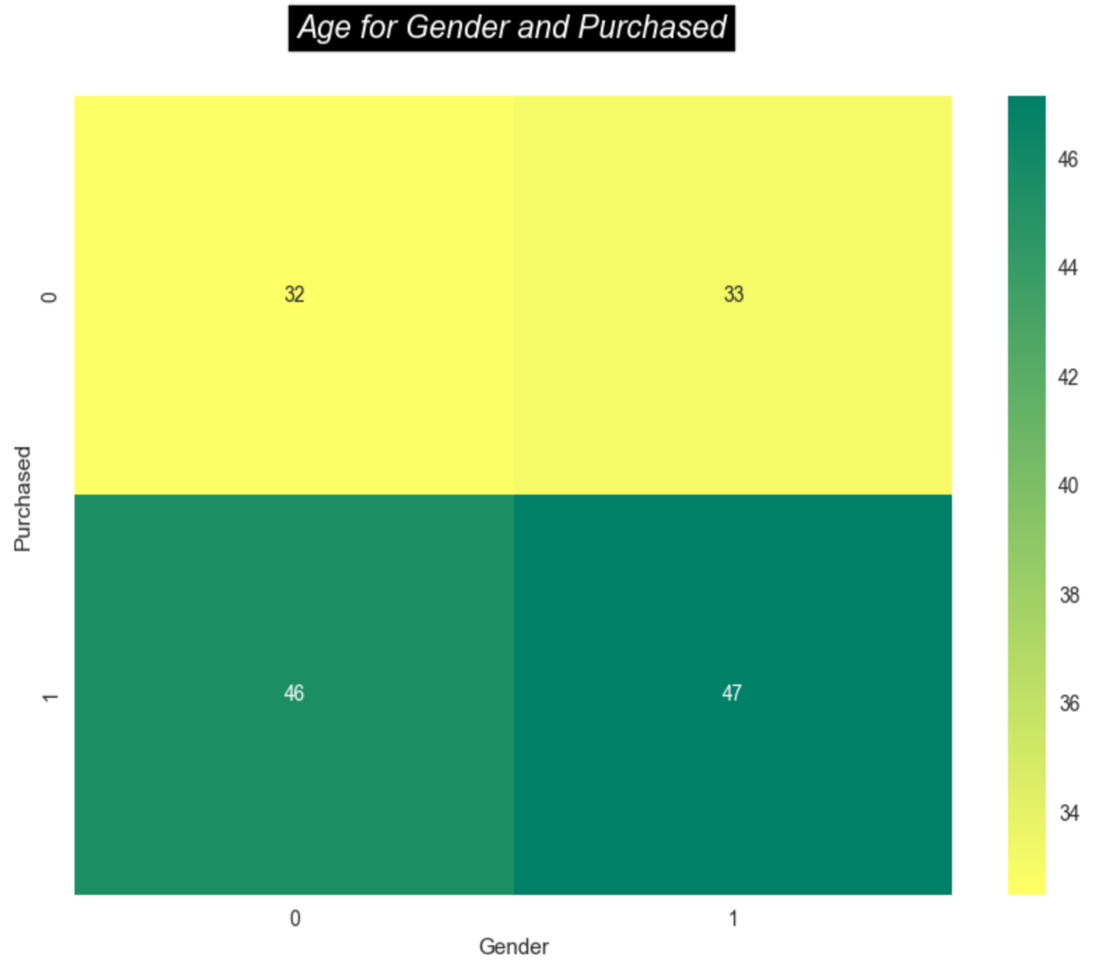
```
# show result in heatmap
# 성별 + 구매 여부에 따른 추정 급여 평균 체크

results = pd.pivot_table(data=df, index='Purchased', columns='Gender', values='EstimatedSalary')

sns.heatmap(results, cmap='summer_r', annot=True)
plt.suptitle('EstimatedSalary for Gender and Purchased', y=1.09, x=0.4, **font)
plt.show()
```



EDA

다변량 분석(성별 + 구매 => 나이 평균)

```
# show result in heatmap
# 성별 + 구매 여부에 따른 나이 평균 체크

results = pd.pivot_table(data=df, index='Purchased', columns='Gender', values='Age')

sns.heatmap(results, cmap='summer_r', annot=True)
plt.suptitle('Age for Gender and Purchased', y=1.09, x=0.4, **font)
plt.show()
```



모델

데이터 스케일 조정(MinMaxScaler)

MinMaxScaler를 통해 급여 데이터 값의 범위를 나이 데이터 값의 범위(18 ~ 60)와 일치시키기

(X-(X의 최솟값) / (X의 최대값-X의 최솟값)

값 * (최대 범위-최소 범위) + 최소범위

(44000 – 15000) / (150000 - 15000) =>

0.2148148148148148 * (60-18) + 18 =>

27.0222222222

```
# standardize EstimatedSalary and Age with MinMaxScaler
df2 = df.copy()
scaler = MinMaxScaler(feature_range=(18,60)).fit(df[['EstimatedSalary']])
df2['EstimatedSalary'] = scaler.transform(df2['EstimatedSalary'].values.reshape(-1,1))
df2


# 스케일러를 통해 데이터의 스케일 조정
# MinMaxScaler => 데이터들이 18~60 사이로 변환
```

	Gender	Age	EstimatedSalary	Purchased
51	1	18	27.022222	0
52	1	29	39.155556	0
53	1	35	20.488889	0
54	1	27	31.377778	0
55	1	24	30.444444	0
56	1	23	28.266667	0
57	0	28	37.911111	0
58	0	22	18.933333	0

400 rows × 4 columns 새 탭에서 열기

모델

나이브 베이즈(naïve bayes)

- 확률 기반 머신러닝 분류 알고리즘
- 모든 특성 값은 서로 독립임을 가정(특성들 사이에 연관 X) => 독립 사건들을 베이즈 이론에 대입시켜 가장 높은 확률의 레이블로 분류를 실행하는 알고리즘

베이즈 이론

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)} \quad \Rightarrow \quad P(A|B)=\frac{P(A\cap B)}{P(B)}$$

P(구매 | 성별, 나이, 급여) => P(성별 | 구매) * P(나이 | 구매) * P(급여 | 구매) * P(구매)

P(구매 X | 성별, 나이, 급여) => P(성별 | 구매 X) * P(나이 | 구매 X) * P(급여 | 구매 X) * P(구매 X)

모델

나이브 베이즈(naïve bayes)

장점
모든 데이터의 특징이 독립적인 사건이라는 가정에도 불구하고 실전에서 높은 정확도를 보임
계산 속도가 다른 모델들에 비해 빠름

단점
모든 데이터의 특징이 독립적인 사건이라는 가정은 일부 분류에는 적합할지 몰라도 다른 분류 모델에는 제약이 될 수 있음

모델

나이브 베이즈

가우시안 나이브 베이즈

특징들의 값들이 정규 분포돼 있다는 가정하에 조건부 확률을 계산
연속적인 성질이 있는 데이터를 분류하는 데 적합

베르누이 나이브 베이즈

이진 데이터(0, 1)

다항 분포 나이브 베이즈

카운트 데이터(나온 횟수 카운트)

주사위 10번
1, 2, 2, 3, 3, 3, 4, 4, 4, 4

베르누이 => (1, 1, 1, 1, 0 ,0)
다항 분포 => (1, 2, 3, 4, 0, 0)

모델

함수 선언(혼동 행렬)

혼동 행렬(오차 행렬)

- 모델의 성능을 평가할 때 사용하는 지표
- 학습된 분류 모델이 예측을 수행하면서 얼마나 헷갈리고 있는지 보여주는 지표
- 예측 오류가 얼마인지 + 어떠한 유형의 예측 오류가 발생하고 있는지

```
def plot_confusion_matrix2(cm, classes,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues):
    """
    This function plots the confusion matrix.
        cm(array): confusion matrix
        classes(dictionary): classes in our target
    """
    plt.figure(figsize=(10,7))
    plt.grid(False)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt ='d'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")


    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.tight_layout()
    plt.show()
```

모델

함수 선언(모델링)

```python
def modeling(x, y, test_size, classes ):

    # split data to train and test
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=0)
    print(20*'-', 'Shape', 20*'-')
    print(f"x_train: {x_train.shape}")
    print(f"y_train: {y_train.shape}")
    print(f"x_test: {x_test.shape}")
    print(f"y_test: {y_test.shape}")

    # define model and fit model
    clf = MultinomialNB()
    clf.fit(x_train, y_train.ravel())

    # prediction and results
    y_pred_test = clf.predict(x_test)

    cm = metrics.confusion_matrix(y_test, y_pred_test)

    acc_test = metrics.accuracy_score(y_test, y_pred_test)

    # Evaluation model
    print('-'*40 , 'Confusion Matrix', '-'*40)
    plot_confusion_matrix2(cm, classes,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues)
    # or use plot_confusion_matrix from sklearn.metrics
    print('-'*20 , 'Accuracy', '-'*20)
    print(f"acc_test: {acc_test}", '\n')

    # print other result about predicted data
    return clf
```

모델 생성/평가

모델링

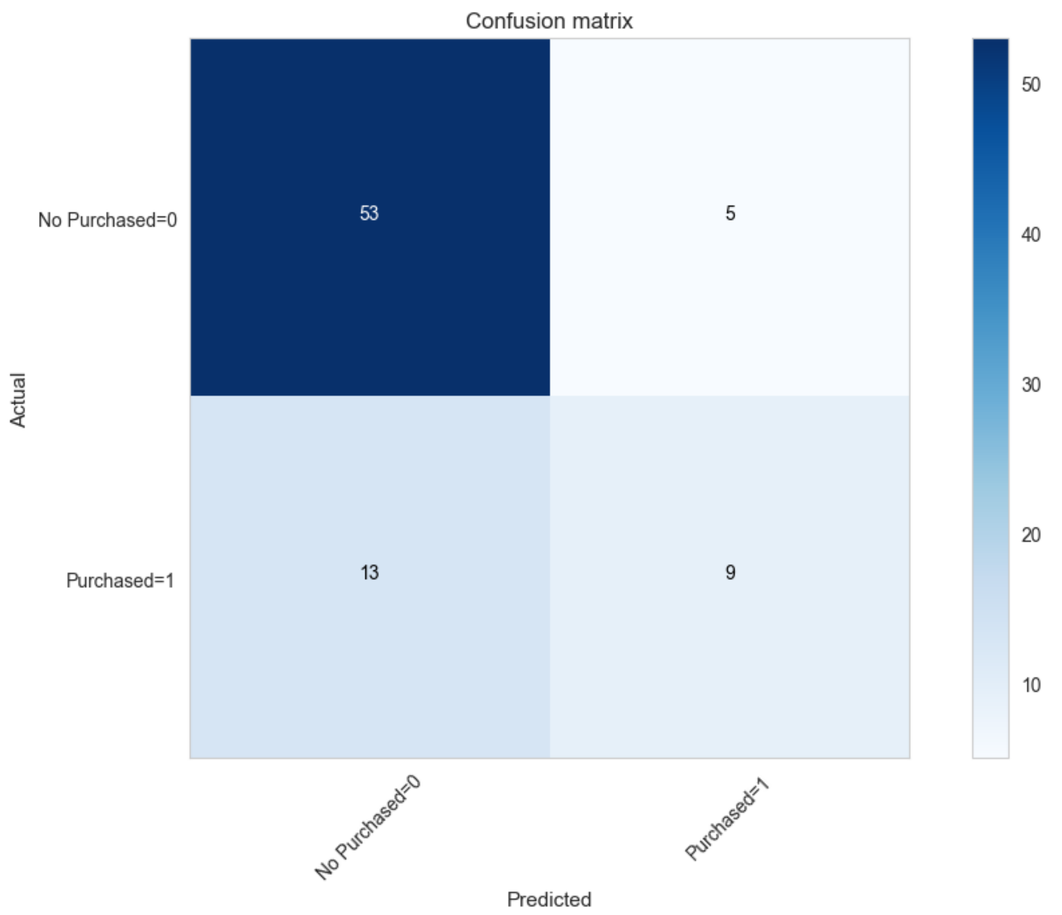
```
# define x (features) and y (target)
x = np.asanyarray(df2.drop('Purchased', axis=1))
y = df2.Purchased.values.reshape(-1,1)
```

```
clf1 = modeling(x, y, 0.2, ['No Purchased=0', 'Purchased=1'])
```

```
-------------------- Shape --------------------
x_train: (320, 3)
y_train: (320, 1)
x_test: (80, 3)
y_test: (80, 1)
```

```
-------------------- Accuracy --------------------
acc_test: 0.775
```

```
---------- Confusion Matrix ----------
```



모델 개선

테스트 비율 변경

30%

```
modeling(x, y, 0.3)
```

```
-------------------- Accuracy --------------------
acc_test: 0.7166666666666667
```

15%

```
modeling(x, y, 0.15)
```

```
-------------------- Accuracy --------------------
acc_test: 0.7666666666666667
```

10%

```
modeling(x, y, 0.1)
```

```
-------------------- Accuracy --------------------
acc_test: 0.775
```

모델 개선

교차 검증(K 폴드)

보편적

테스트 데이터 : 전체 데이터셋의 20%
학습 데이터 : 나머지 데이터의 90%
검증 데이터 : 나머지 데이터의 10%

테스트 데이터 : 전체 데이터셋의 20%
학습 데이터 : 전체 데이터셋의 80%

단점

1. 데이터가 충분하지 않을 경우 10%도 검증 데이터로 할당하기 아까움
2. 한쪽에 편중된 데이터

모델 개선

교차 검증(K 폴드)

K-폴드 교차 검증 1

학습 데이터와 테스트 데이터로

모든 데이터를 k개의 폴드 세트로 만든 뒤, k-1 폴드는 학습, 제외된 폴드는 테스트(검증)용으로 사용되는 방식

K-폴드 교차 검증 2

학습 데이터의 일정 부분을 검증 데이터로

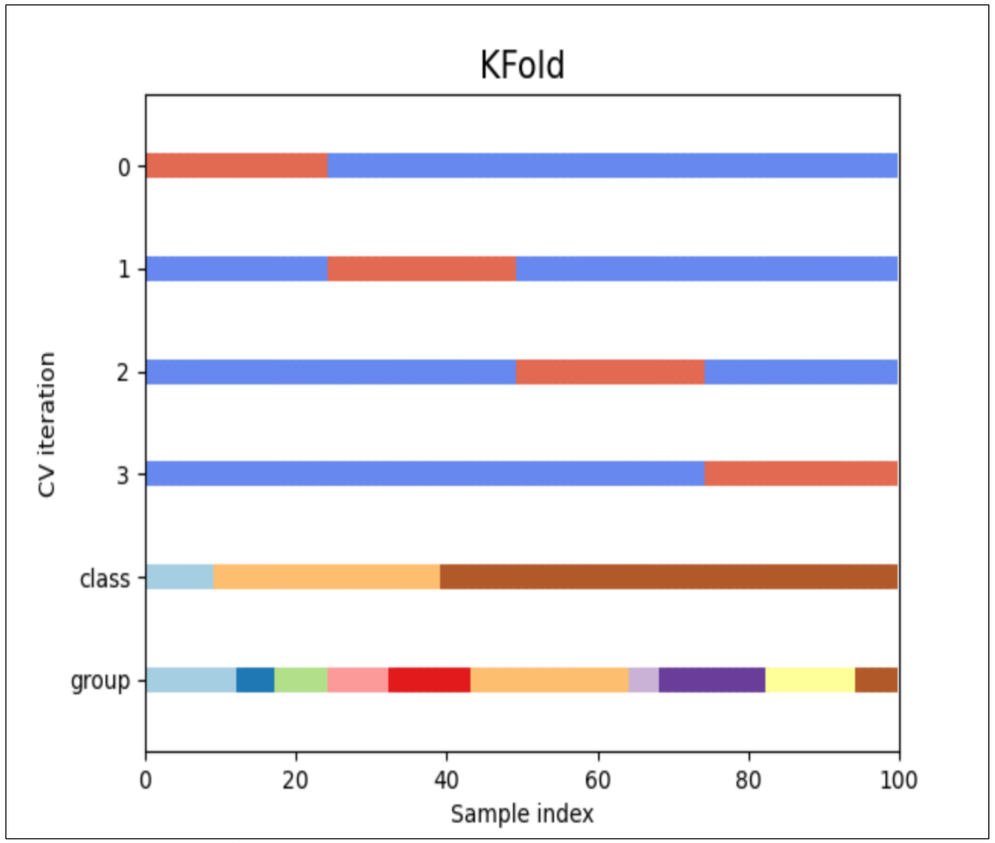
n번의 검증 과정을 통해 학습 데이터의 모든 데이터를 한 번씩 검증 데이터로 사용해서 평균낸 값을 평가 지표로 사용하는 방식

장점

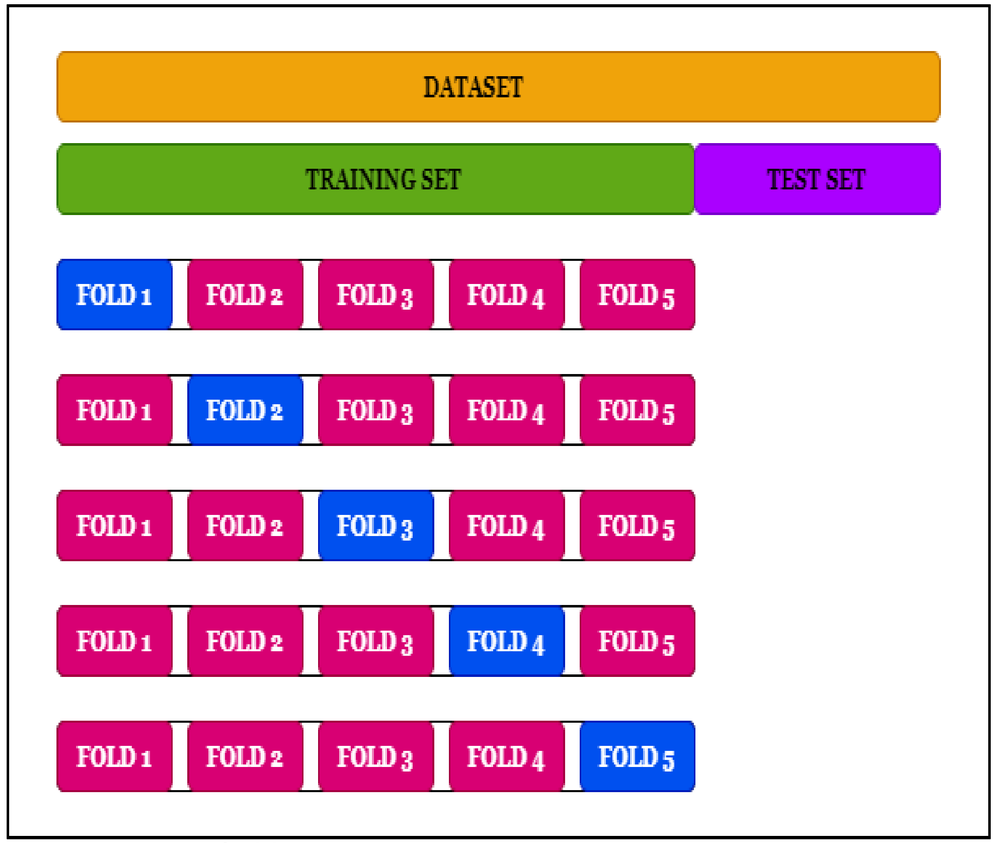
1. 검증 결과가 일정 데이터에 치우치지 않고 모든 데이터에 대한 결과이므로 신뢰성이 높음
2. 검증 데이터를 따로 분리하지 않아도 됨

모델 개선

교차 검증(K 폴드)



출처 : https://scikit-learn.org(kfold)



출처 : https://codong.tistory.com/37

모델 개선

교차 검증(K 폴드)

```python
def Perform_cross_val(model, k, x, y, scoring):
    """
    perform cross validation
        model: logistic model
        k(scaler): the value for n_splits in KFold()
        x(DataFrame or array):  x_train
        y(DataFrame or array): y_train
        scoring(string): an approach for evaluation in cross validation
    """

    kf = KFold(n_splits=k)
    cv_results = cross_val_score(model, x, y, cv=kf, scoring=scoring)
    cv_mean = np.mean(cv_results)
    print('-'*20, f"CV for k={k}, scoring={scoring}", '-'*20)
    print(f"CV mean: {cv_mean}")
    print(f"CV results: {cv_results}\n")

# 교차 검증(K 폴드 교차 검증)
```

```python
Perform_cross_val(clf1, 10, x, y, scoring='accuracy')
```

```
-------------------- CV for k=10, scoring=accuracy --------------------
CV mean: 0.6399999999999999
CV results: [0.925 0.825 0.8   0.7   0.85  0.4   0.5   0.55  0.6   0.25 ]
```

모델 개선

교차 검증(K 폴드)

```python
def find_fold_index(k, x):
    """
    Find fold index in kfold
        k(scaler): the value used for n_splits in KFold()
        x(DataFrame or array): x_train
    """

    my_fold_index = []
    j=1
    for _ , test in KFold(k).split(x):

        my_fold_index = []
        for i in test:
            my_fold_index.append(i)
        print(f"fold {j}: [{my_fold_index[0]},{my_fold_index[-1]}]")
        j += 1
```

```python
# first find index of datapoint in fold 1 and 3 by following func:
find_fold_index(10, x)
```

```
fold 1: [0,39]
fold 2: [40,79]
fold 3: [80,119]
fold 4: [120,159]
fold 5: [160,199]
fold 6: [200,239]
fold 7: [240,279]
fold 8: [280,319]
fold 9: [320,359]
fold 10: [360,399]
```

fold6, fold10 삭제

```python
# delete above index from df2
df3 = df2.drop(df2.index[np.r_[200:239+1, 360:399+1]], axis=0)
print(f'df2:{df2.shape}')
print(f'df3:{df3.shape}')
```

```
df2:(400, 4)
df3:(320, 4)
```

최종 모델 생성/평가

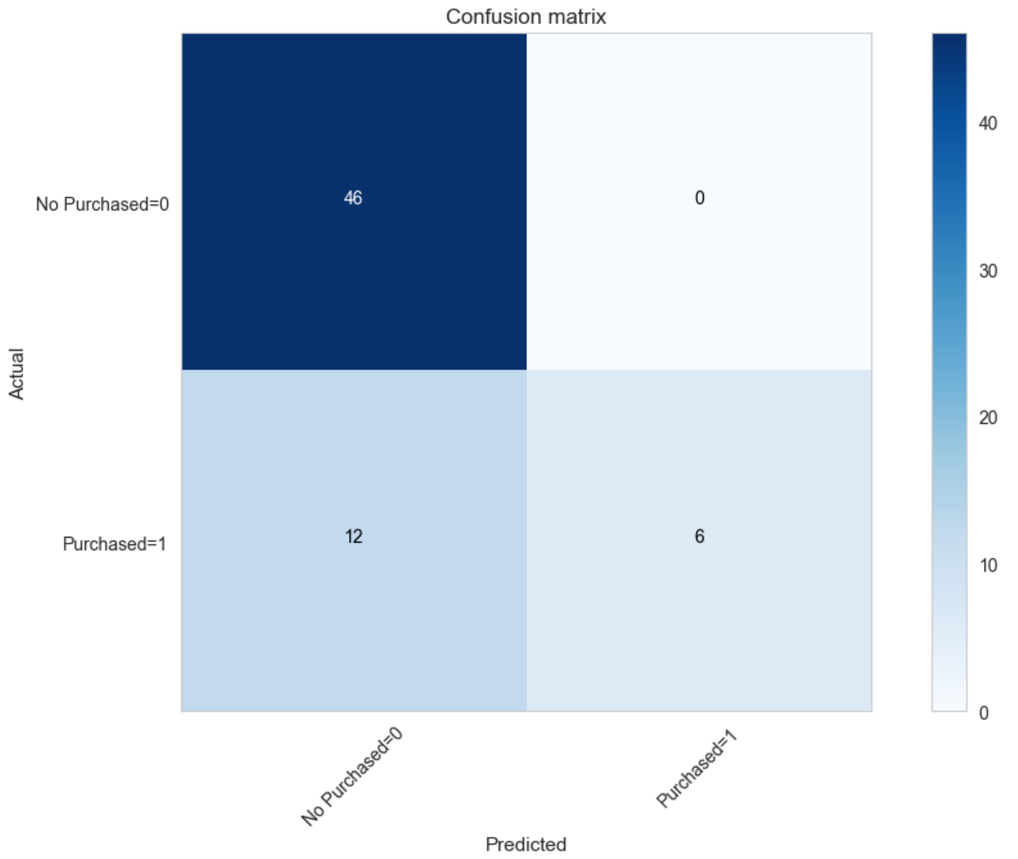
모델링

```
# define new x and y
x2 = np.asanyarray(df3.drop('Purchased', axis=1))
y2 = df3.Purchased.values.reshape(-1,1)
```

```
clf2 = modeling(x2, y2, 0.2)
```

```
-------------------- Shape --------------------
x_train: (256, 3)
y_train: (256, 1)
x_test: (64, 3)
y_test: (64, 1)


------------------- Accuracy -------------------
acc_test: 0.8125
```

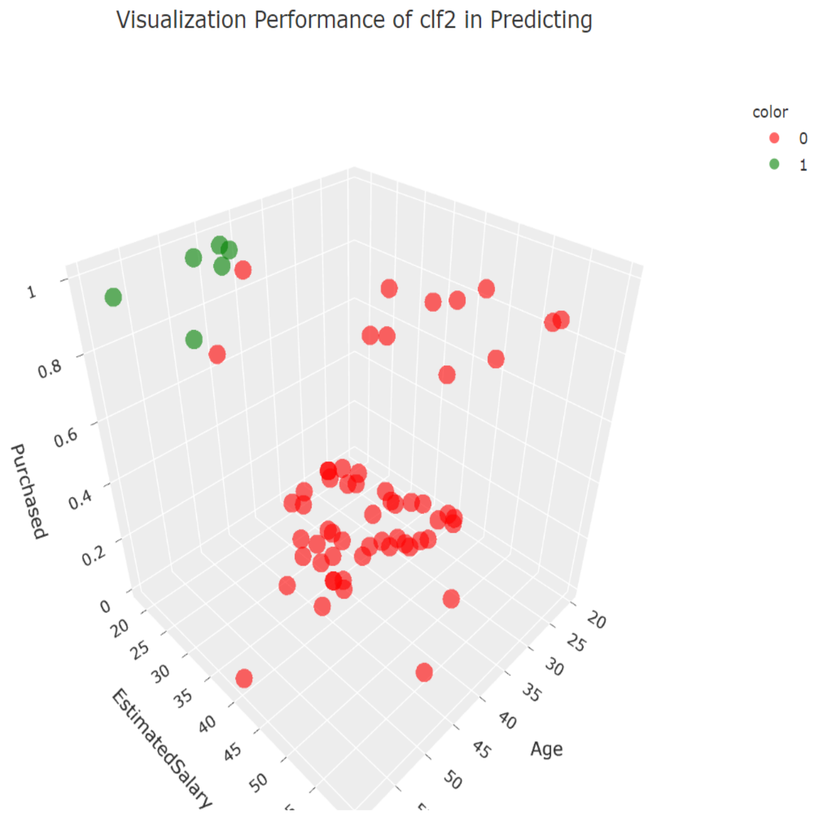
```
------------------------------------ Confusion Matrix ------------------------------------
```



최종 모델

시각화

```
%matplotlib inline
x_train, x_test, y_train, y_test = train_test_split(x2, y2, test_size=0.2, random_state=0)
x_test = np.concatenate((x_test, y_test), axis=1)
x_test = pd.DataFrame(x_test, columns=['Gender', 'Age', 'EstimatedSalary', 'Purchased'])

fig = px.scatter_3d(
    data_frame= x_test,
    x=x_test.Age,
    y=x_test.EstimatedSalary,
    z=x_test.Purchased,
    color=clf2.predict(x_test.drop('Purchased', axis=1)).astype(str),
    color_discrete_sequence={0:'red', 1:'green'},
    template='ggplot2',
    opacity=0.6,
    height=700,
    title=f'Visualization Performance of clf2 in Predicting')

pio.show(fig)
```



- 빨간색 : 구매하지 않을 것을 예상
- 초록색 : 구매 예상
- 구매하지 않을 거라 생각했는데 구매한 사람이 많음(나이가 적고 급여가 높은 사람들)

테스트(결과)

학습, 예측

```
# fit final model on all of data (train + test)
final_model = MultinomialNB()
final_model.fit(x2, y2)

# 예측할 데이터
new_data = {'Gender': [0], 'Age': [23], 'EstimatedSalary': 20000}

new_sample = pd.DataFrame(new_data)
print(f"Gender: {new_sample['Gender'].values[0]}\n"
      f"Age: {new_sample['Age'].values[0]}\n"
      f"EstimatedSalary: {new_sample['EstimatedSalary'].values[0]}")

# 예측할 데이터 스케일러
new_sample['EstimatedSalary'] = scaler.transform(new_sample[['EstimatedSalary']])
ns = np.asanyarray(new_sample)

# 예측 결과
result = final_model.predict(ns)
print('='*38)
print(f"class {result}")
```

```
Gender: 0
Age: 23
EstimatedSalary: 20000
======================================
class [0]
```

나이 ▼
급여 ▼

```
Gender: 0
Age: 23
EstimatedSalary: 150000
======================================
class [0]
```

나이 ▼
급여 ▲

```
Gender: 0
Age: 48
EstimatedSalary: 20000
======================================
class [1]
```

나이 ▲
급여 ▼

```
Gender: 0
Age: 48
EstimatedSalary: 150000
======================================
class [0]
```

나이 ▲
급여 ▲

결론

나이가 어리거나 급여가 높은 경우 예측 정확도가 떨어지는 것으로 보임

사용한 데이터셋이 400명의 고객 정보만 가지고 있어서 많이 부족한 것 같다고 느낌

소감

수학이 많이 중요하다는 것을 알게 되었다.

발표를 준비하면서 새로운 내용들을 많이 접하게 되었는데,
아직 지식이 부족해서 완벽하게 이해하지 못 한 것 같아서 아쉬웠다.

참고문헌

https://scikit-learn.org/

허민석. (2020). 나의 첫 머신러닝/딥러닝. 위키북스.

권철민. (2020). 파이썬 머신러닝 완벽 가이드. 위키북스.

감사합니다