캐글 코리아 4차 대회

학습내용

- 상위 솔루션을 분석해 봅니다.(1st)

- 대회 링크 : https://www.kaggle.com/c/kakr-4th-competition/overview (https://www.kaggle.com/c/kakr-4th-competition/overview)
- 참고 링크 : https://www.kaggle.com/code/bestend/kakr-4th-1st-place-solution (https://www.kaggle.com/code/bestend/kakr-4th-1st-place-solution) (1st)

목차

01. 데이터 준비 및 라이브러리 임포트

목차로 이동하기

설치

- pip install [라이브러리명]

In [67]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import lightgbm as lgb
import pycaret

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import warnings
warnings.filterwarnings('ignore')

from IPython.display import display

pd.options.display.max_rows = 10000
pd.options.display.max_columns = 1000
pd.options.display.max_colwidth = 1000
```

In [68]:

```
print("lightgbm ver : ", lgb.__version__)
print("pycaret ver : ", pycaret.__version__)
```

```
lightgbm ver :  3.1.1
pycaret ver :  2.3.10
```

데이터 탐색

데이터 정보

```
age : 나이
workclass : 고용 형태
fnlwgt : 사람 대표성을 나타내는 가중치 (final weight의 약자)
education : 교육 수준 (최종 학력)
education_num : 교육 수준 수치
marital_status: 결혼 상태
occupation : 업종
relationship : 가족 관계
race : 인종
sex : 성별
capital_gain : 양도 소득
capital_loss : 양도 손실
hours_per_week : 주당 근무 시간
native_country : 국적
income : 수익 (예측해야 하는 값, target variable)
```

In [69]:

```
dirname = "data/4th_kaggle"

train = pd.read_csv(os.path.join(dirname, 'train.csv'), index_col='id')
test = pd.read_csv(os.path.join(dirname, 'test.csv'), index_col='id')

train.shape, test.shape
```

Out[69]:

```
((26049, 15), (6512, 14))
```

EDA 참고 링크

[https://github.com/Aditya-Mankar/Census-Income-Prediction (https://github.com/Aditya-Mankar/Census-Income-Prediction)](https://github.com/Aditya-Mankar/Census-Income-Prediction)

02. 데이터 탐색

[목차로 이동하기](#)

In [4]:

```
# 데이터 살펴보기
train.head()
```

Out[4]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
id								
0	40	Private	168538	HS-grad	9	Married-civ-spouse	Sales	Husband
1	17	Private	101626	9th	5	Never-married	Machine-op-inspct	Own-child
2	18	Private	353358	Some-college	10	Never-married	Other-service	Own-child
3	21	Private	151158	Some-college	10	Never-married	Prof-specialty	Own-child
4	24	Private	122234	Some-college	10	Never-married	Adm-clerical	Not-in-family

In [5]:

```
# 데이터 구조
print('Rows: {} Columns: {}'.format(train.shape[0], train.shape[1]))
print('Rows: {} Columns: {}'.format(test.shape[0], test.shape[1]))
```

```
Rows: 26049 Columns: 15
Rows: 6512 Columns: 14
```

In [6]:

```
### 데이터 정보 확인
train.info(), test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26049 entries, 0 to 26048
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             26049 non-null  int64 
 1   workclass       26049 non-null  object
 2   fnlwgt          26049 non-null  int64 
 3   education       26049 non-null  object
 4   education_num   26049 non-null  int64 
 5   marital_status  26049 non-null  object
 6   occupation      26049 non-null  object
 7   relationship    26049 non-null  object
 8   race            26049 non-null  object
 9   sex             26049 non-null  object
 10  capital_gain    26049 non-null  int64 
 11  capital_loss    26049 non-null  int64 
 12  hours_per_week  26049 non-null  int64 
 13  native_country  26049 non-null  object
 14  income          26049 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.2+ MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6512 entries, 0 to 6511
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             6512 non-null   int64 
 1   workclass       6512 non-null   object
 2   fnlwgt          6512 non-null   int64 
 3   education       6512 non-null   object
 4   education_num   6512 non-null   int64 
 5   marital_status  6512 non-null   object
 6   occupation      6512 non-null   object
 7   relationship    6512 non-null   object
 8   race            6512 non-null   object
 9   sex             6512 non-null   object
 10  capital_gain    6512 non-null   int64 
 11  capital_loss    6512 non-null   int64 
 12  hours_per_week  6512 non-null   int64 
 13  native_country  6512 non-null   object
dtypes: int64(6), object(8)
memory usage: 763.1+ KB
```

Out[6]:

```
(None, None)
```

In [7]:

```
### 통계적 요약
display(train.describe().T)
display(test.describe().T)
```

	count	mean	std	min	25%	50%	75%
age	26049.0	38.569235	13.671489	17.0	28.0	37.0	48.0
fnlwgt	26049.0	190304.481708	105966.299073	13769.0	118108.0	178866.0	237735.0
education_num	26049.0	10.088372	2.567610	1.0	9.0	10.0	12.0
capital_gain	26049.0	1087.689700	7388.854690	0.0	0.0	0.0	0.0
capital_loss	26049.0	87.732734	403.230205	0.0	0.0	0.0	0.0
hours_per_week	26049.0	40.443126	12.361850	1.0	40.0	40.0	45.0

	count	mean	std	min	25%	50%	75%
age	6512.0	38.631296	13.516418	17.0	28.00	37.0	48.00
fnlwgt	6512.0	187673.824939	103849.326430	12285.0	116504.25	176882.0	235850.75
education_num	6512.0	10.049908	2.593033	1.0	9.00	10.0	12.00
capital_gain	6512.0	1037.483876	7371.453668	0.0	0.00	0.0	0.00
capital_loss	6512.0	85.588145	401.904741	0.0	0.00	0.0	0.00
hours_per_week	6512.0	40.414773	12.290491	1.0	40.00	40.0	45.00

In [8]:

```
# null 값이 존재하는지 확인
dat = (train.isnull().sum() / train.shape[0]) * 100
round(dat, 2).astype(str) + ' %'
```

Out[8]:

```
age               0.0 %
workclass         0.0 %
fnlwgt            0.0 %
education         0.0 %
education_num     0.0 %
marital_status    0.0 %
occupation        0.0 %
relationship      0.0 %
race              0.0 %
sex               0.0 %
capital_gain      0.0 %
capital_loss      0.0 %
hours_per_week    0.0 %
native_country    0.0 %
income            0.0 %
dtype: object
```

In [9]:

```
dat = (train.isin(['?']).sum() / train.shape[0])
round(dat, 2).astype(str) + ' %'
```

Out[9]:

```
age                0.0 %
workclass         0.06 %
fnlwgt             0.0 %
education          0.0 %
education_num      0.0 %
marital_status     0.0 %
occupation        0.06 %
relationship       0.0 %
race               0.0 %
sex                0.0 %
capital_gain       0.0 %
capital_loss       0.0 %
hours_per_week     0.0 %
native_country    0.02 %
income             0.0 %
dtype: object
```

In [10]:

```
dat = (test.isin(['?']).sum() / train.shape[0])
round(dat, 2).astype(str) + ' %'
```

Out[10]:

```
age                0.0 %
workclass         0.01 %
fnlwgt             0.0 %
education          0.0 %
education_num      0.0 %
marital_status     0.0 %
occupation        0.01 %
relationship       0.0 %
race               0.0 %
sex                0.0 %
capital_gain       0.0 %
capital_loss       0.0 %
hours_per_week     0.0 %
native_country     0.0 %
dtype: object
```

In [11]:

```
# 소득 비율
income = train['income'].value_counts(normalize=True)
round(income * 100, 2).astype('str') + ' %'
```

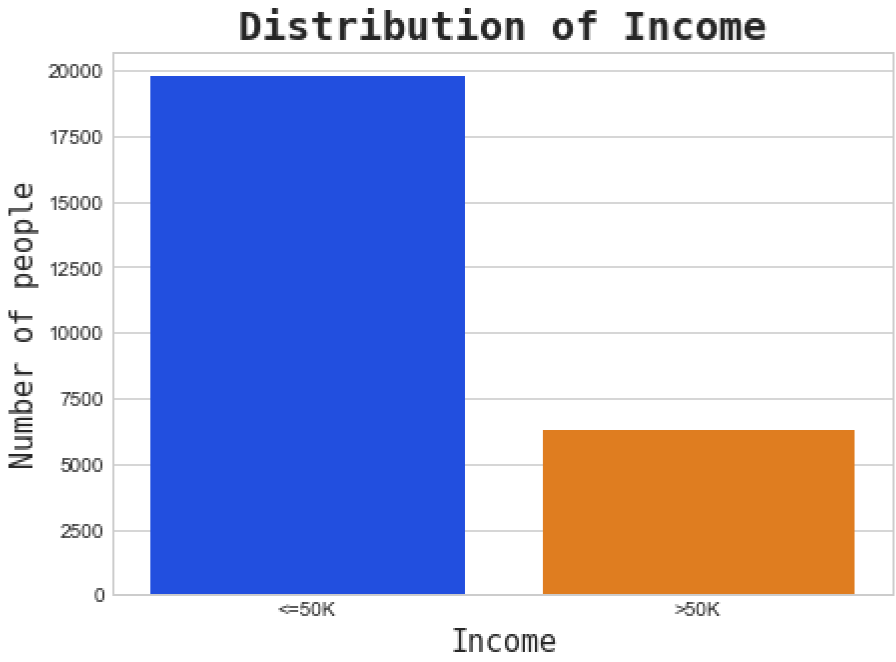
Out[11]:

```
<=50K    75.8 %
>50K     24.2 %
Name: income, dtype: object
```

데이터 탐색 및 분석

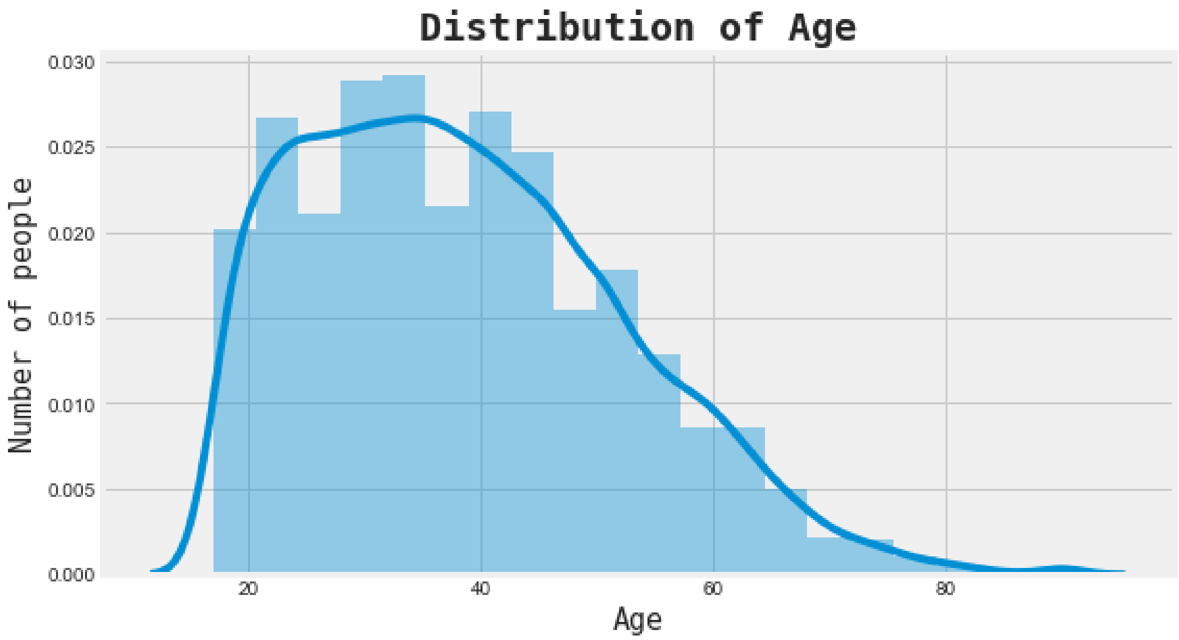
In [12]:

```
# Creating a barplot for 'Income'
income = train['income'].value_counts()

plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(7, 5))
sns.barplot(income.index, income.values, palette='bright')

plt.title('Distribution of Income',
          fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt.xlabel('Income', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=10)
plt.show()
```



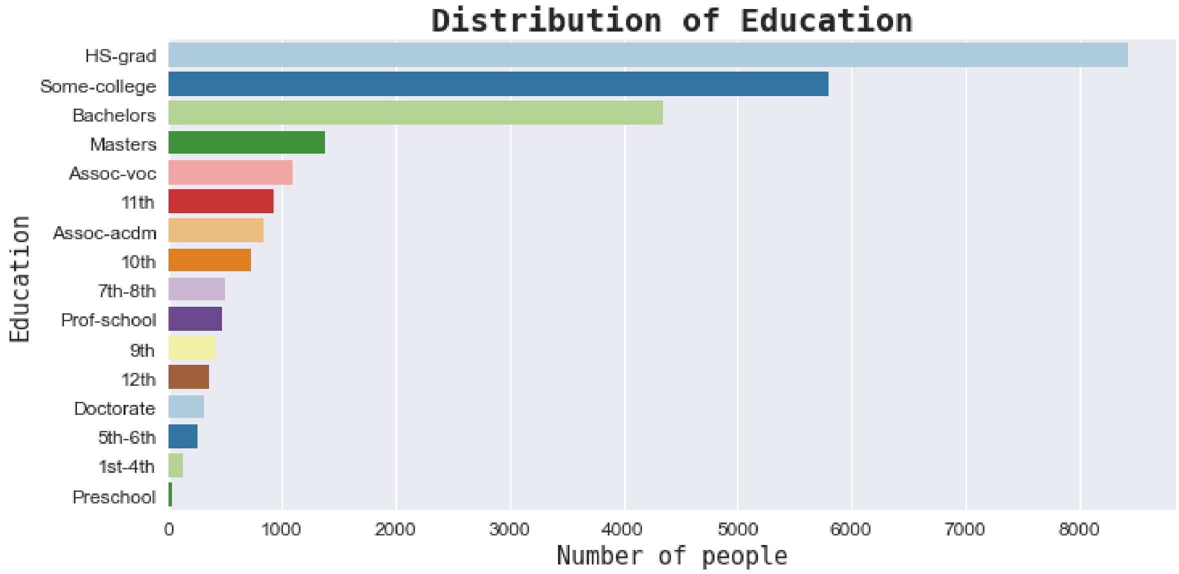
In [13]:

```
# Creating a distribution plot for 'Age'
age = train['age'].value_counts()

plt.figure(figsize=(10, 5))
plt.style.use('fivethirtyeight')
sns.distplot(train['age'], bins=20)

plt.title('Distribution of Age', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=10)
plt.show()
```



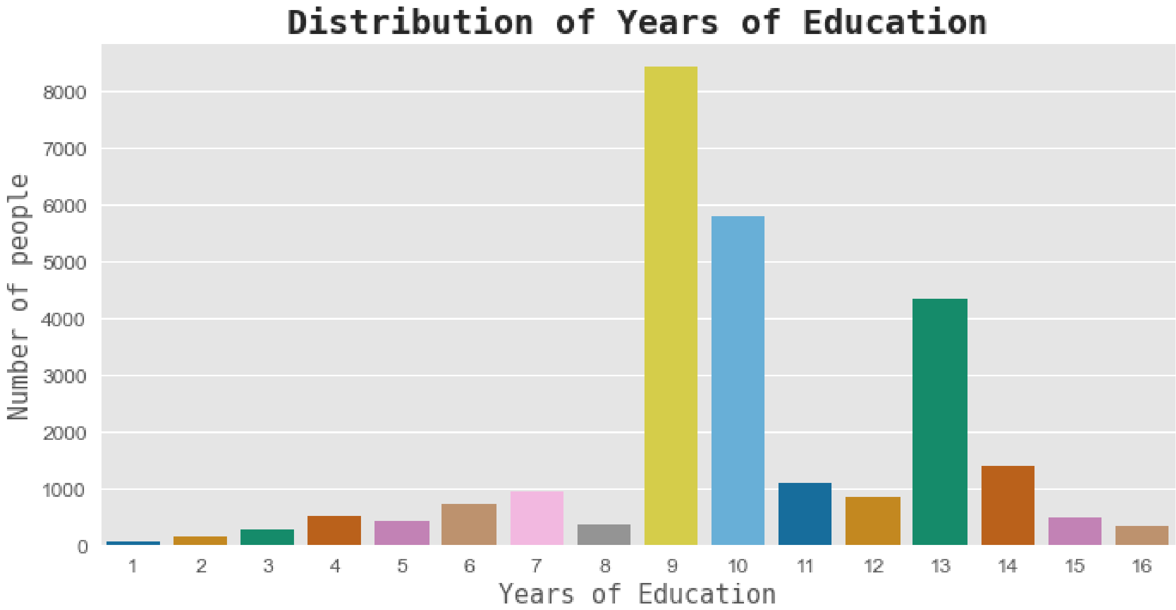
In [14]:

```
# Creating a barplot for 'Education'
edu = train['education'].value_counts()

plt.style.use('seaborn')
plt.figure(figsize=(10, 5))
sns.barplot(edu.values, edu.index, palette='Paired')

plt.title('Distribution of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Education', fontdict={'fontname': 'Monospace',
                                  'fontsize': 15})
plt.tick_params(labelsize=12)
plt.show()
```



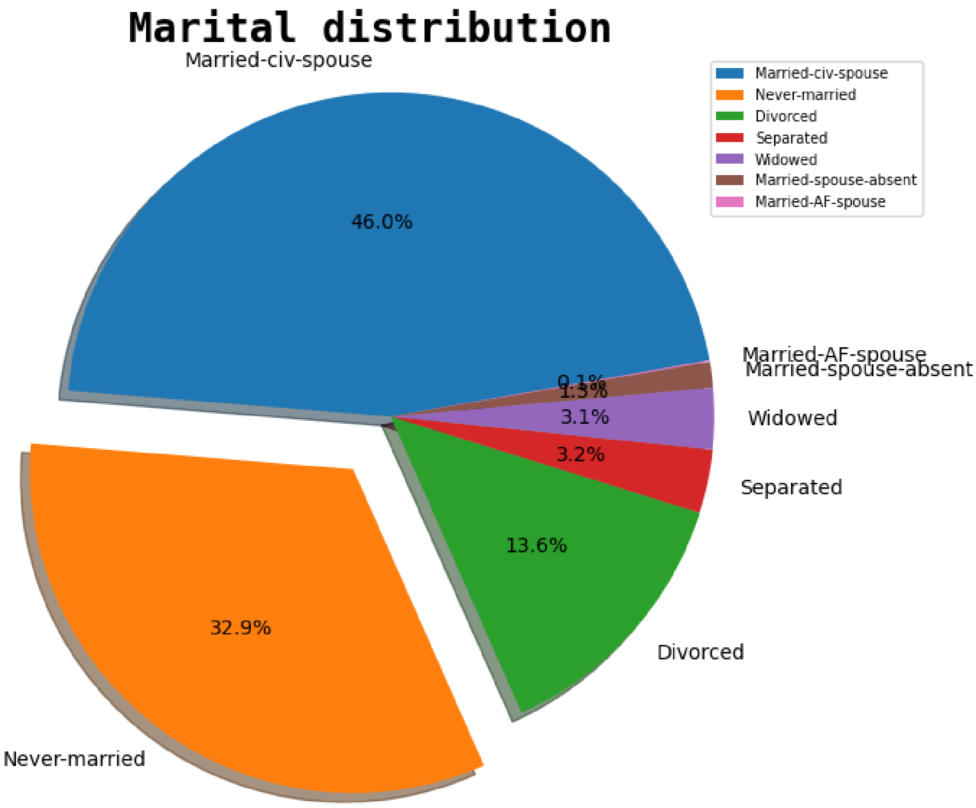
In [15]:

```
# Creating a barplot for 'Years of Education'
edu_num = train['education_num'].value_counts()

plt.style.use('ggplot')
plt.figure(figsize=(10, 5))
sns.barplot(edu_num.index, edu_num.values, palette='colorblind')

plt.title('Distribution of Years of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Years of Education', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.show()
```



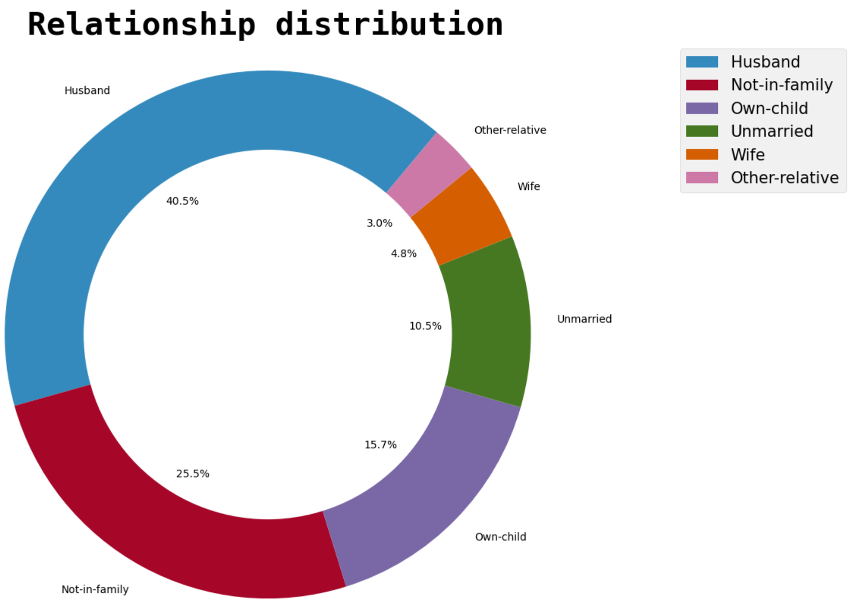
In [16]:

```
# Creating a pie chart for 'Marital status'
marital = train['marital_status'].value_counts()

plt.style.use('default')
plt.figure(figsize=(10, 7))
plt.pie(marital.values,
        labels=marital.index, startangle=10,
        explode=(0, 0.20, 0, 0, 0, 0, 0), shadow=True, autopct='%1.1f%%')

plt.title('Marital distribution',
          fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt.legend()
plt.legend(prop={'size': 7})
plt.axis('equal')
plt.show()
```



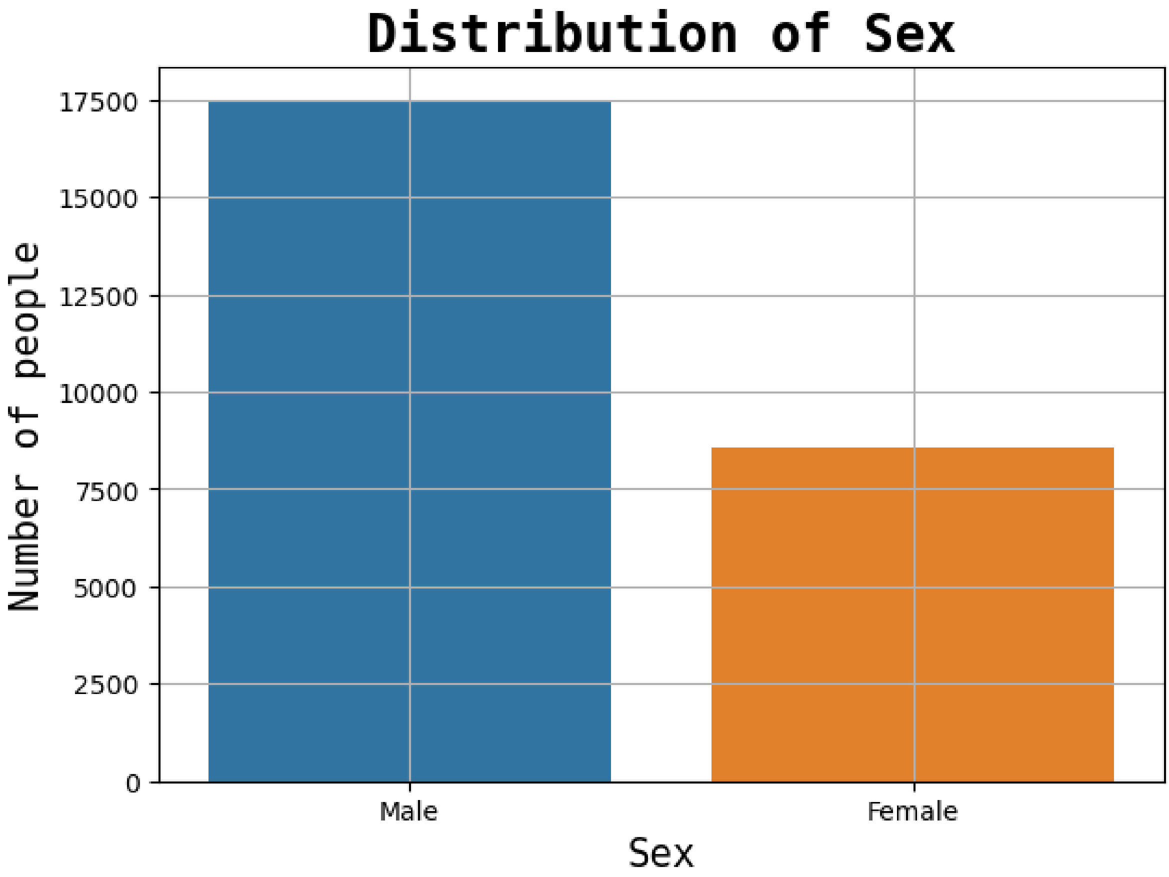
In [17]:

```
# Creating a donut chart for 'Age'
relation = train['relationship'].value_counts()

plt.style.use('bmh')
plt.figure(figsize=(20, 10))
plt.pie(relation.values,
        labels=relation.index,
        startangle=50, autopct='%1.1f%%')

centre_circle = plt.Circle((0, 0), 0.7, fc='white')

fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Relationship distribution',
          fontdict={
          'fontname': 'Monospace', 'fontsize': 30, 'fontweight': 'bold'})
plt.axis('equal')
plt.legend(prop={'size': 15})
plt.show()
```



In [18]:

```
# Creating a barplot for 'Sex'
sex = train['sex'].value_counts()

plt.style.use('default')
plt.figure(figsize=(7, 5))
sns.barplot(sex.index, sex.values)

plt.title('Distribution of Sex', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Sex', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=10)
plt.grid()
plt.show()
```

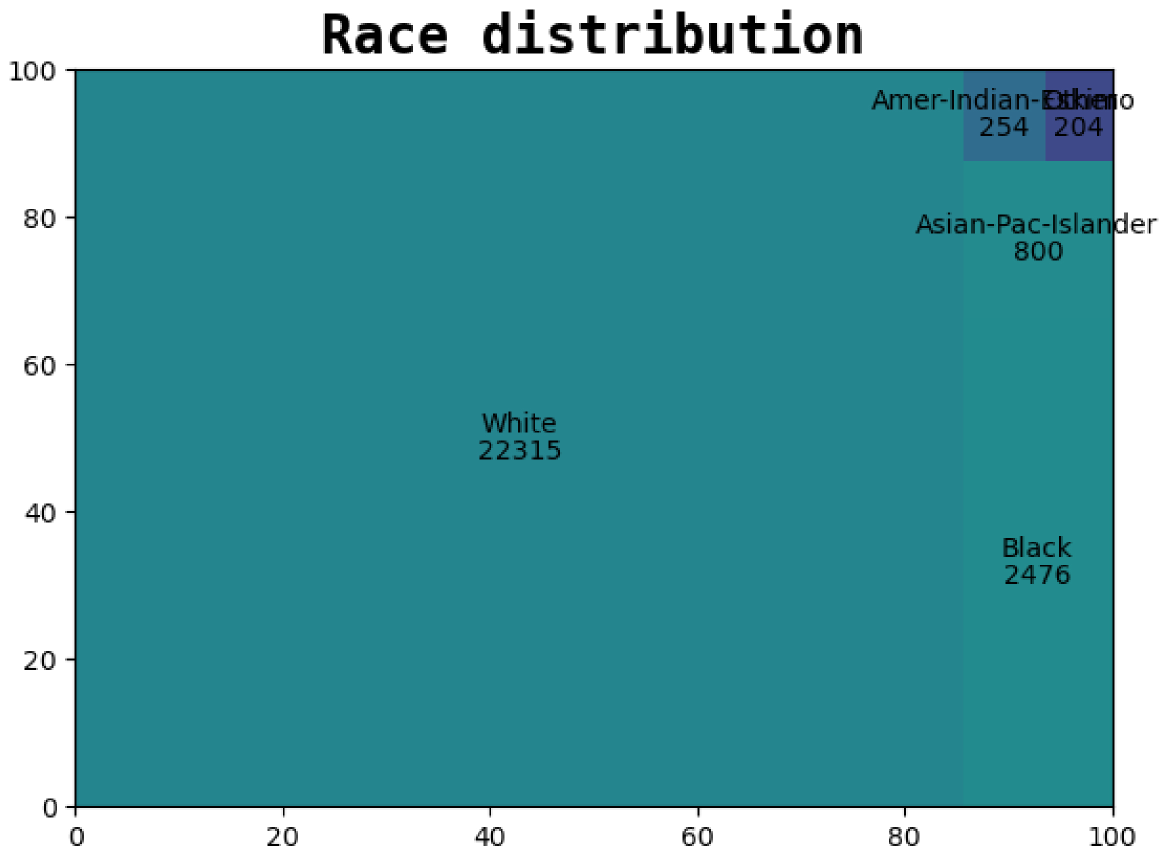


In [19]:

```
# Creating a Treemap for 'Race'
import squarify
```

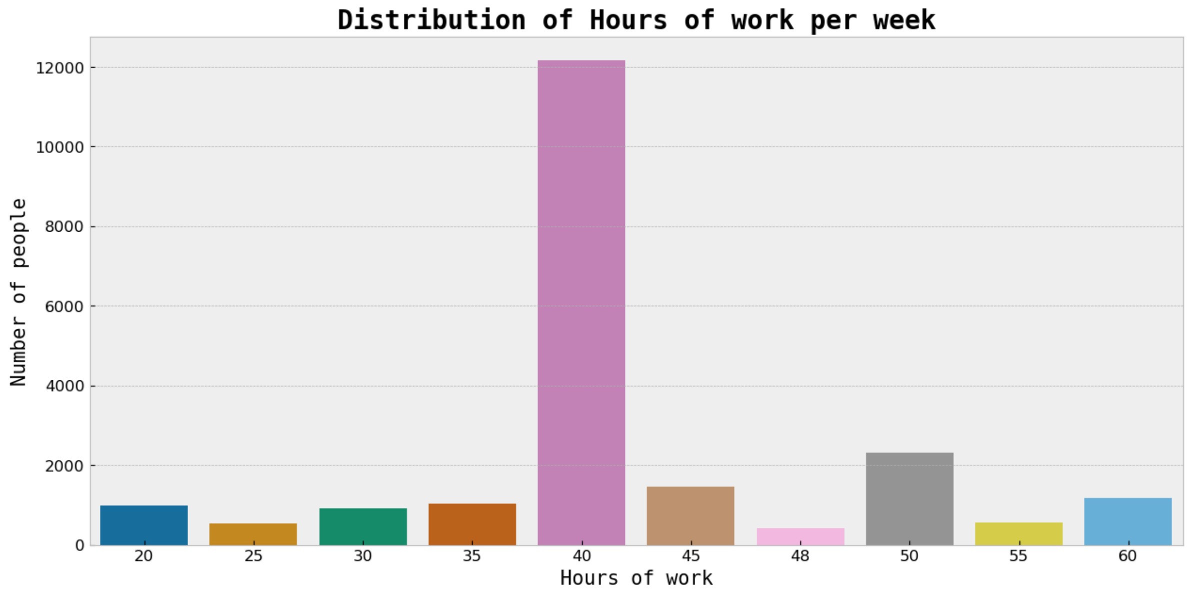
In [20]:

```
race = train['race'].value_counts()

plt.style.use('default')
plt.figure(figsize=(7, 5))
squarify.plot(sizes=race.values, label=race.index, value=race.values)

plt.title('Race distribution', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.show()
```



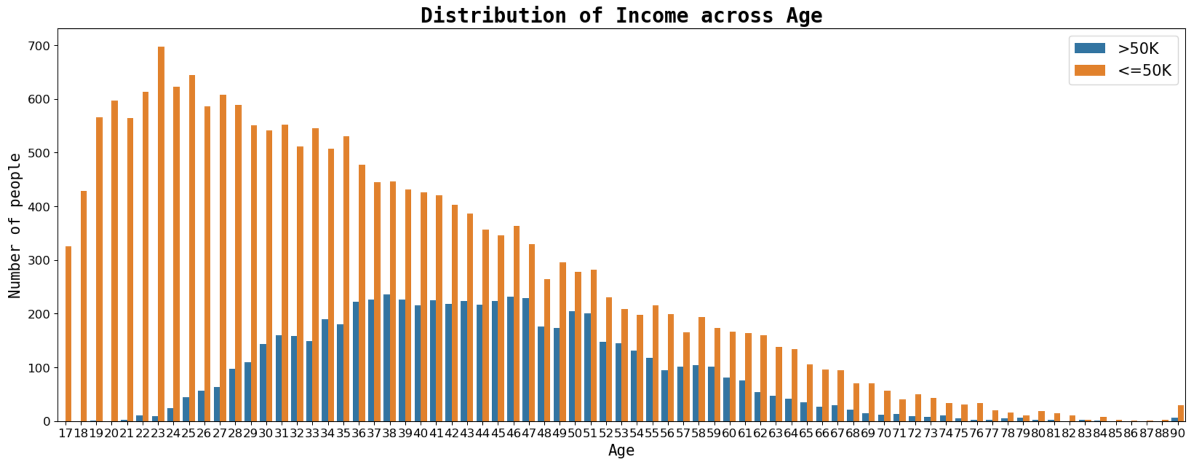
In [21]:

```
# Creating a barplot for 'Hours per week'
hours = train['hours_per_week'].value_counts().head(10)

plt.style.use('bmh')
plt.figure(figsize=(15, 7))
sns.barplot(hours.index, hours.values, palette='colorblind')

plt.title('Distribution of Hours of work per week', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Hours of work', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.show()
```



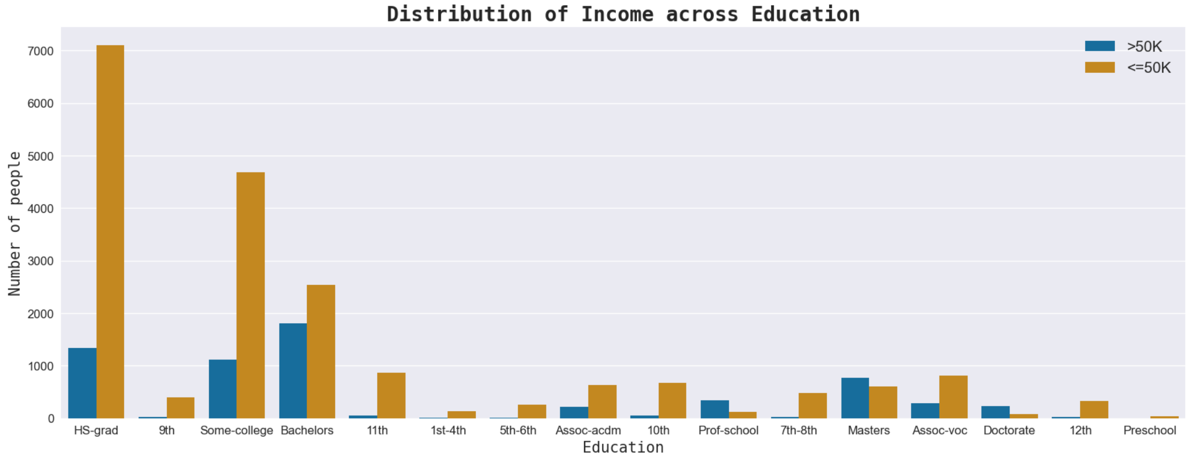
In [22]:

```
# Creating a countplot of income across age
plt.style.use('default')
plt.figure(figsize=(20, 7))
sns.countplot(train['age'], hue=train['income'])

plt.title('Distribution of Income across Age', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```



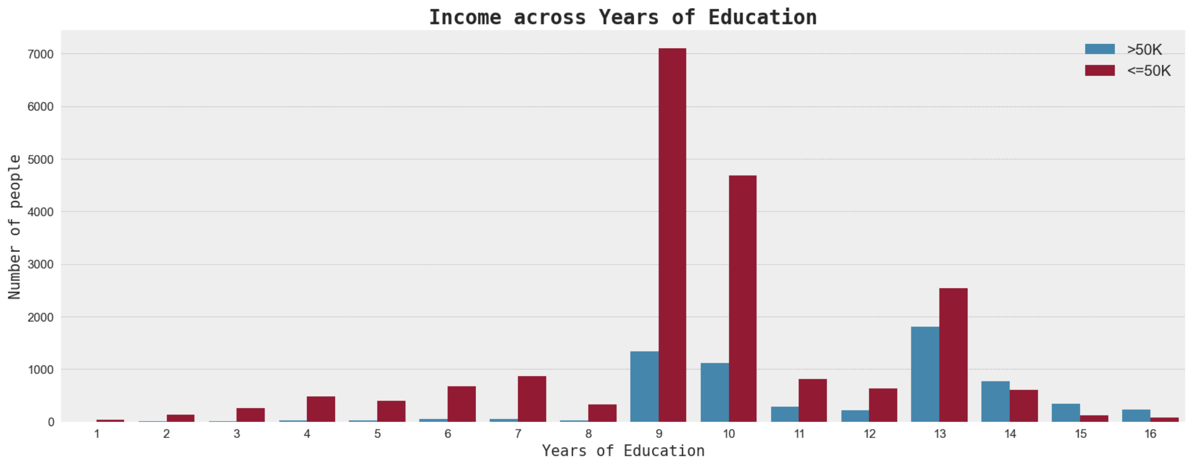
In [23]:

```
# Creating a countplot of income across education
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(train['education'],
              hue=train['income'],
              palette='colorblind')

plt.title('Distribution of Income across Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Education', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```



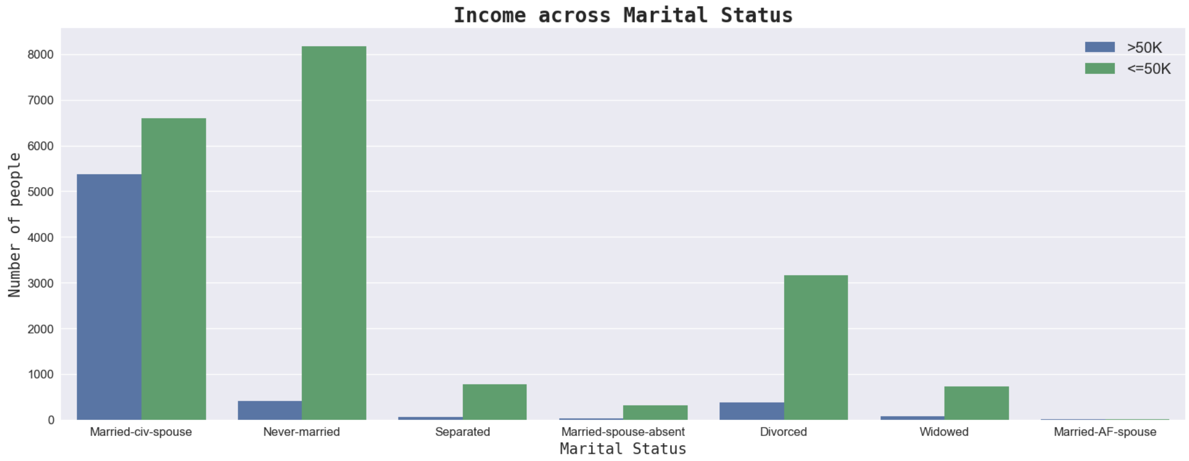
In [24]:

```
# Creating a countplot of income across years of education
plt.style.use('bmh')
plt.figure(figsize=(20, 7))
sns.countplot(train['education_num'],
              hue=train['income'])

plt.title('Income across Years of Education', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Years of Education', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.savefig('bi2.png')
plt.show()
```



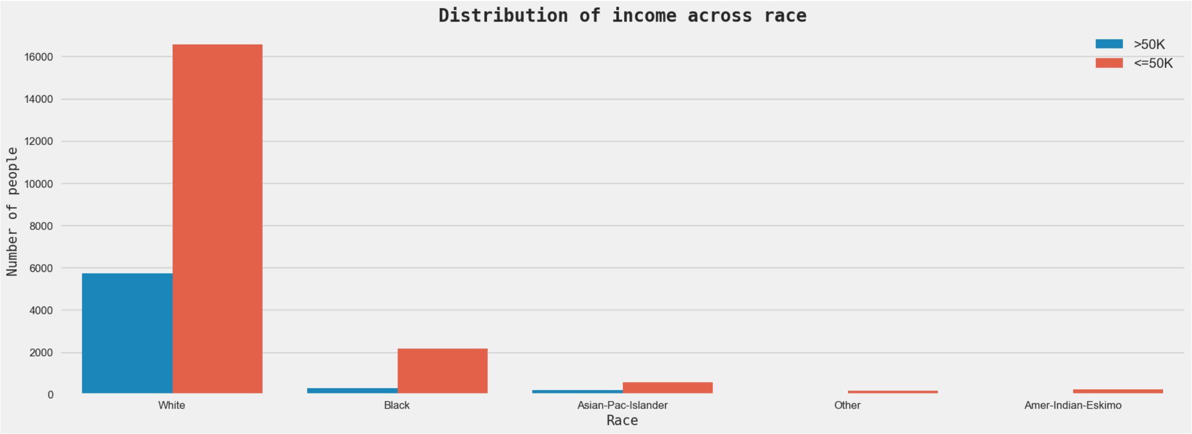
In [25]:

```
# Creating a countplot of income across Marital Status
plt.style.use('seaborn')
plt.figure(figsize=(20, 7))
sns.countplot(train['marital_status'], hue=train['income'])

plt.title('Income across Marital Status', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Marital Status', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```



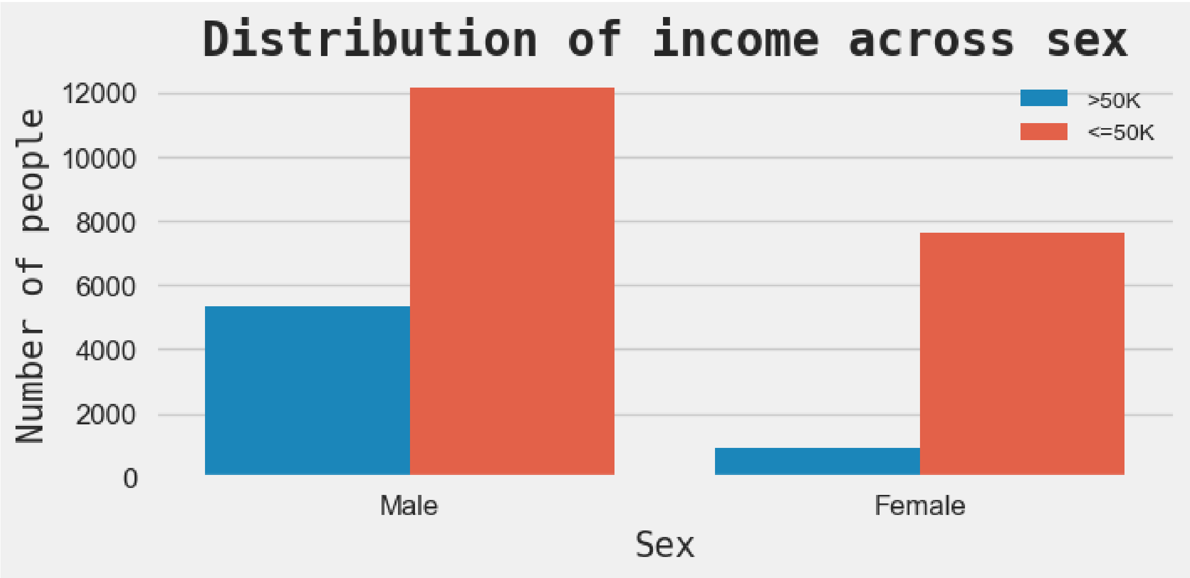
In [26]:

```
# Creating a countplot of income across race
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20, 7))
sns.countplot(train['race'], hue=train['income'])

plt.title('Distribution of income across race', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Race', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 15})
plt.show()
```



In [27]:

```
# Creating a countplot of income across sex
plt.style.use('fivethirtyeight')
plt.figure(figsize=(7, 3))
sns.countplot(train['sex'], hue=train['income'])

plt.title('Distribution of income across sex', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Sex', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={
           'fontname': 'Monospace', 'fontsize': 15})

plt.tick_params(labelsize=12)
plt.legend(loc=1, prop={'size': 10})
plt.savefig('bi3.png')
plt.show()
```

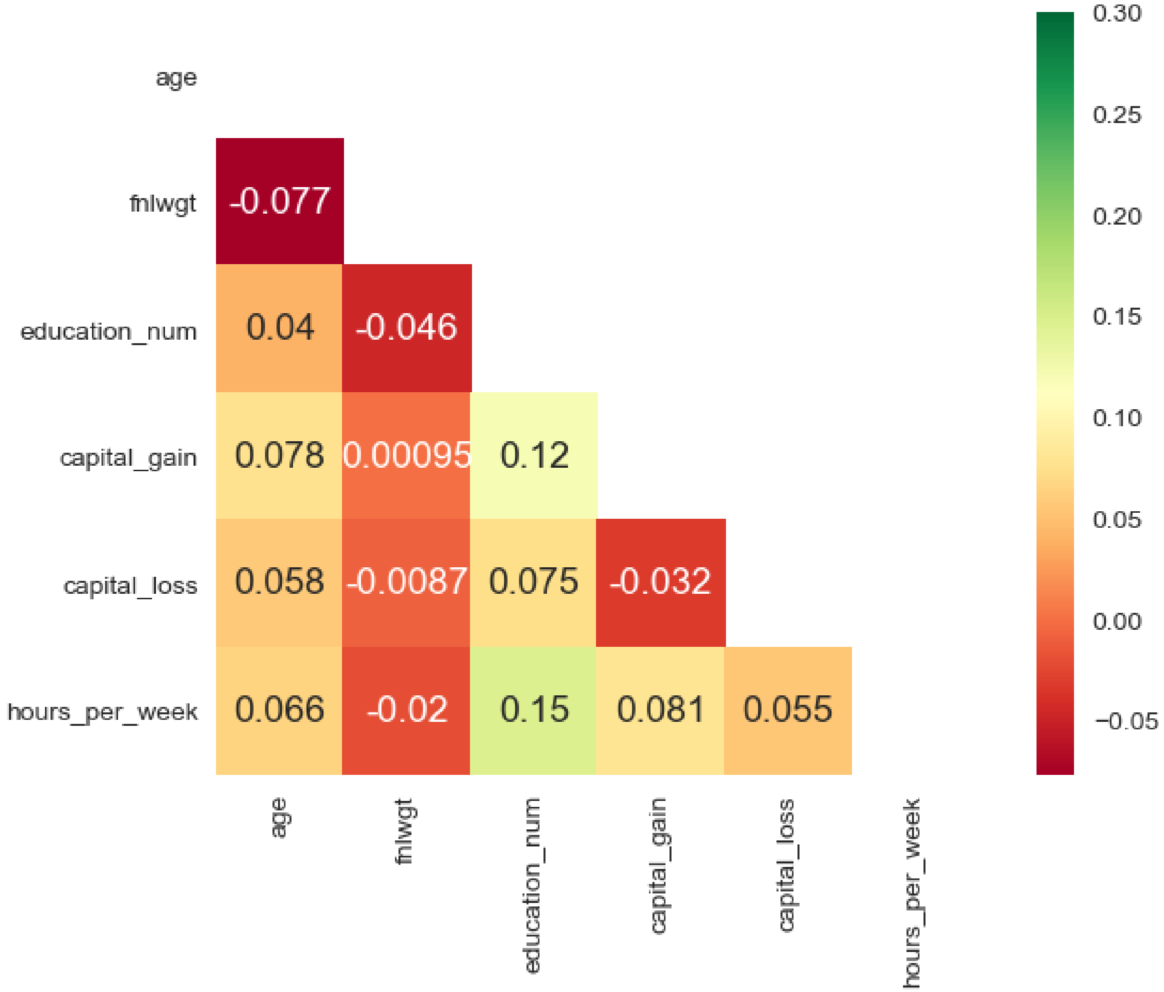


In [28]:

```
# Creating a pairplot of dataset
sns.pairplot(train)
plt.savefig('multi1.png')
plt.show()
```

In [30]:

```
corr = train.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(7, 5))
    ax = sns.heatmap(corr, mask=mask, vmax=.3, square=True,
                     annot=True, cmap='RdYlGn')
plt.savefig('multi2.png')
plt.show()
```



03. 모델링 - pycaret를 활용

[목차로 이동하기](#)

PyCaret은 무엇일까?

- PyCaret은 Python의 오픈 소스 로우 코드 머신러닝 라이브러리로, ML 실험에서 가설을 인사이트주기 시간으로 줄이는 것을 목표로합니다.

- 이를 통해 데이터 과학자는 종단 간 실험을 빠르고 효율적으로 수행 할 수 있습니다.
- 다른 오픈 소스 기계 학습 라이브러리와 비교하여 PyCaret은 코드 몇 줄만으로 복잡한 기계 학습 작업을 수행하는 데 사용할 수 있는 대체 로우 코드 라이브러리입니다.
- PyCaret은 간단하고 사용하기 쉽습니다.
- PyCaret에서 수행되는 모든 작업 은 배포를 위해 완전히 조정 된 사용자 지정 파이프 라인 에 자동으로 저장됩니다.
- PyCaret은 본질적으로 scikit-learn, XGBoost, LightGBM, spaCy 등과 같은 여러 기계 학습 라이브러리 및 프레임 워크를 둘러싼 Python 래퍼입니다.

- [https://pycaret.org/ (https://pycaret.org/)](https://pycaret.org/)

설치 및 작업 순서

- pip install pycaret

순서

- 01 setup module를 사용하여 setup
- 02 데이터를 지정하고 preprocessing을 적용.(setup)
- 03 필요시 모델 추가. 모델 확인 후, 커스텀 모델을 추가하거나 모델 튜닝 - add_metric, create_models
- 04 모델 학습 및 비교. compare_models()
- 05 실험 로그를 찍고, 여러가지로 확인

In [70]:

```python
import pycaret

print(pycaret.__version__)
```

```
2.3.10
```

In [71]:

```python
# pycaret 초기화
from pycaret.classification import *

clf = setup(data=train,
            target='income',
            session_id=999,
            high_cardinality_features=['native_country'],
            use_gpu=False,
            silent=True,
            fix_imbalance=False,
            normalize=True,
            feature_selection=False)
```

	Description	Value
0	session_id	999
1	Target	income
2	Target Type	Binary
3	Label Encoded	<=50K: 0, >50K: 1
4	Original Data	(26049, 15)
5	Missing Values	False
6	Numeric Features	5
7	Categorical Features	9
8	Ordinal Features	False
9	High Cardinality Features	True
10	High Cardinality Method	frequency
11	Transformed Train Set	(18234, 64)
12	Transformed Test Set	(7815, 64)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	2901
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant

	Description	Value
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	True
30	Normalize Method	zscore
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Remove Perfect Collinearity	True
45	Clustering	False
46	Clustering Iteration	None
47	Polynomial Features	False
48	Polynomial Degree	None
49	Trignometry Features	False
50	Polynomial Threshold	None
51	Group Features	False
52	Feature Selection	False
53	Feature Selection Method	classic
54	Features Selection Threshold	None
55	Feature Interaction	False
56	Feature Ratio	False
57	Interaction Threshold	None
58	Fix Imbalance	False
59	Fix Imbalance Method	SMOTE

In [72]:

```
# compare_models 함수로 15개의 기본 모델을 학습하고 성능 비교 가능
best_5 = compare_models(n_select=5, sort='F1')
best_5
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.8727	0.9259	0.6495	0.7839	0.7102	0.6296	0.6343	3.9980
lightgbm	Light Gradient Boosting Machine	0.8705	0.9243	0.6500	0.7750	0.7069	0.6246	0.6287	0.2450
xgboost	Extreme Gradient Boosting	0.8674	0.9218	0.6468	0.7653	0.7010	0.6166	0.6202	5.9620
ada	Ada Boost Classifier	0.8597	0.9138	0.6165	0.7553	0.6787	0.5902	0.5952	0.5140
gbc	Gradient Boosting Classifier	0.8631	0.9187	0.5847	0.7912	0.6724	0.5883	0.5991	1.7380
rf	Random Forest Classifier	0.8547	0.9031	0.6153	0.7368	0.6705	0.5782	0.5822	1.3770
lr	Logistic Regression	0.8524	0.9062	0.5991	0.7373	0.6610	0.5679	0.5730	0.7980
svm	SVM - Linear Kernel	0.8483	0.0000	0.5624	0.7450	0.6398	0.5463	0.5556	0.1080
et	Extra Trees Classifier	0.8323	0.8788	0.6014	0.6681	0.6329	0.5246	0.5259	1.7020
lda	Linear Discriminant Analysis	0.8406	0.8926	0.5649	0.7125	0.6300	0.5302	0.5361	0.1720
knn	K Neighbors Classifier	0.8310	0.8539	0.5900	0.6683	0.6265	0.5179	0.5197	2.5570
dt	Decision Tree Classifier	0.8130	0.7469	0.6194	0.6096	0.6143	0.4910	0.4911	0.1150
ridge	Ridge Classifier	0.8395	0.0000	0.5063	0.7442	0.6025	0.5065	0.5213	0.0380
nb	Naive Bayes	0.5641	0.8128	0.9491	0.3501	0.5115	0.2470	0.3511	0.0380
qda	Quadratic Discriminant Analysis	0.4942	0.5509	0.6602	0.2738	0.3853	0.0699	0.0895	0.1180
dummy	Dummy Classifier	0.7596	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0280

Out[72]:

```
[<catboost.core.CatBoostClassifier at 0x7f8c5280b370>,
 LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_by
tree=1.0,
                importance_type='split', learning_rate=0.1, max_depth
=-1,
                min_child_samples=20, min_child_weight=0.001, min_spl
it_gain=0.0,
                n_estimators=100, n_jobs=-1, num_leaves=31, objective
=None,
                random_state=999, reg_alpha=0.0, reg_lambda=0.0, sile
nt=True,
```

```
               subsample=1.0, subsample_for_bin=200000, subsample_fr
eq=0),
 XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
               colsample_bylevel=1, colsample_bynode=1, colsample_byt
ree=1,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, gamma=0, gpu_id=-1, grow_policy='dep
thwise',
               importance_type=None, interaction_constraints='',
               learning_rate=0.300000012, max_bin=256, max_cat_to_one
hot=4,
               max_delta_step=0, max_depth=6, max_leaves=0, min_child
_weight=1,
               missing=nan, monotone_constraints='()', n_estimators=1
00,
               n_jobs=-1, num_parallel_tree=1, objective='binary:logi
stic',
               predictor='auto', random_state=999, reg_alpha=0, ...),
 AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learnin
g_rate=1.0,
                    n_estimators=50, random_state=999),
 GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
init=None,
                            learning_rate=0.1, loss='deviance', max_d
epth=3,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_s
plit=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimator
s=100,
                            n_iter_no_change=None, presort='deprecate
d',
                            random_state=999, subsample=1.0, tol=0.00
01,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False)]
```

사용 가능한 모델

- [https://pycaret.gitbook.io/docs/ (https://pycaret.gitbook.io/docs/)](https://pycaret.gitbook.io/docs/)

In [73]:

```
# 지정 모델에서 성능 좋은 5가지 모델을 추출
best_5 = compare_models(n_select=5,
            include=['lightgbm', 'xgboost', 'gbc', 'rf', 'ada', 'catboost', 'et'])
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.8727	0.9259	0.6495	0.7839	0.7102	0.6296	0.6343	3.6070
lightgbm	Light Gradient Boosting Machine	0.8705	0.9243	0.6500	0.7750	0.7069	0.6246	0.6287	0.2720
xgboost	Extreme Gradient Boosting	0.8674	0.9218	0.6468	0.7653	0.7010	0.6166	0.6202	6.6840
gbc	Gradient Boosting Classifier	0.8631	0.9187	0.5847	0.7912	0.6724	0.5883	0.5991	1.8350
ada	Ada Boost Classifier	0.8597	0.9138	0.6165	0.7553	0.6787	0.5902	0.5952	0.5630
rf	Random Forest Classifier	0.8547	0.9031	0.6153	0.7368	0.6705	0.5782	0.5822	1.5600
et	Extra Trees Classifier	0.8323	0.8788	0.6014	0.6681	0.6329	0.5246	0.5259	1.8330

In [74]:

```
### 5개의 모델을 앙상블하여 성능 개선
blended = blend_models(estimator_list=best_5, fold=5, method='auto')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8783	0.9289	0.6587	0.7992	0.7222	0.6451	0.6501
1	0.8711	0.9256	0.6431	0.7822	0.7059	0.6244	0.6293
2	0.8634	0.9214	0.6135	0.7719	0.6836	0.5980	0.6044
3	0.8684	0.9269	0.6271	0.7824	0.6962	0.6135	0.6196
4	0.8697	0.9232	0.6279	0.7868	0.6984	0.6167	0.6231
Mean	0.8702	0.9252	0.6340	0.7845	0.7013	0.6195	0.6253
Std	0.0048	0.0027	0.0155	0.0088	0.0127	0.0154	0.0149

In [75]:

```
blended
```

Out[75]:

```
VotingClassifier(estimators=[('catboost',
                              <catboost.core.CatBoostClassifier object
at 0x7f8c3743e070>),
                             ('lightgbm',
                              LGBMClassifier(boosting_type='gbdt',
                                             class_weight=None,
                                             colsample_bytree=1.0,
                                             importance_type='split',
                                             learning_rate=0.1, max_de
pth=-1,
                                             min_child_samples=20,
                                             min_child_weight=0.001,
                                             min_split_gain=0.0,
                                             n_estimators=100, n_jobs=
-1,
                                             num_leaves=31, objec...
                                                         min_weight_fr
action_leaf=0.0,
                                                         n_estimators=
100,
                                                         n_iter_no_cha
nge=None,
                                                         presort='depr
ecated',
                                                         random_state=
999,
                                                         subsample=1.
0,
                                                         tol=0.0001,
                                                         validation_fr
action=0.1,
                                                         verbose=0,
                                                         warm_start=Fa
lse)),
                             ('ada',
                              AdaBoostClassifier(algorithm='SAMME.R',
                                                 base_estimator=None,
                                                 learning_rate=1.0,
                                                 n_estimators=50,
                                                 random_state=999))],
                 flatten_transform=True, n_jobs=-1, verbose=False,
                 voting='soft', weights=None)
```

In [81]:

```
# 앞서 kfold로 훈련했을때 가장 best였던 parameter를 기준으로
# train data를 전체 다사용해서 최종 학습
final = finalize_model(blended)
final
```

Out[81]:

```
VotingClassifier(estimators=[('catboost',
                              <catboost.core.CatBoostClassifier object
at 0x7f8c52815f40>),
                             ('lightgbm',
                              LGBMClassifier(boosting_type='gbdt',
                                             class_weight=None,
                                             colsample_bytree=1.0,
                                             importance_type='split',
                                             learning_rate=0.1, max_de
pth=-1,
                                             min_child_samples=20,
                                             min_child_weight=0.001,
                                             min_split_gain=0.0,
                                             n_estimators=100, n_jobs=
-1,
                                             num_leaves=31, objec...
                                                         min_weight_fr
action_leaf=0.0,
                                                         n_estimators=
100,
                                                         n_iter_no_cha
nge=None,
                                                         presort='depr
ecated',
                                                         random_state=
999,
                                                         subsample=1.
0,
                                                         tol=0.0001,
                                                         validation_fr
action=0.1,
                                                         verbose=0,
                                                         warm_start=Fa
lse)),
                             ('ada',
                              AdaBoostClassifier(algorithm='SAMME.R',
                                                 base_estimator=None,
                                                 learning_rate=1.0,
                                                 n_estimators=50,
                                                 random_state=999))],
                 flatten_transform=True, n_jobs=-1, verbose=False,
                 voting='soft', weights=None)
```

In [77]:

```
test.columns
```

Out[77]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_count
ry'],
      dtype='object')
```

In [82]:

```
test.head()
```

Out[82]:

education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss
10	Never-married	Adm-clerical	Other-relative	White	Female	0	0
9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
10	Never-married	Handlers-cleaners	Own-child	White	Male	0	0
11	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
16	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0

In [83]:

```
# test set에 대해서 모델 평가
# Label라는 컬럼이 생기고, 여기
prediction_test = predict_model(final, data=test)
prediction_test
```

Out[83]:

marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
Never-married	Adm-clerical	Other-relative	White	Female	0	0	40
Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	50
Never-married	Handlers-cleaners	Own-child	White	Male	0	0	25
Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	50
Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	99
...	...	...	...	...	...	...	...
Married-civ-spouse	Sales	Husband	White	Male	0	0	40
Married-civ-spouse	Tech-support	Husband	White	Male	0	0	40
Married-civ-spouse	Other-service	Husband	White	Male	0	0	40
Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40
Divorced	Handlers-cleaners	Unmarried	White	Female	0	0	36

In [91]:

```
prediction_test.loc[ prediction_test['Label'] == "<=50K", "pred" ] = 0
prediction_test.loc[ prediction_test['Label'] == ">50K", "pred" ] = 1
prediction_test['pred'] = prediction_test['pred'].astype(int)
prediction_test['pred']
```

Out[91]:

```
id
0       0
1       1
2       0
3       1
4       0
       ..
6507    0
6508    1
6509    0
6510    0
6511    0
Name: pred, Length: 6512, dtype: int64
```

In [93]:

```
submission = pd.read_csv(os.path.join(dirname, 'sample_submission.csv'))
display(submission.head(5))
submission['prediction'] = prediction_test['pred']
submission
```

	id	prediction
0	0	0
1	1	0
2	2	0
3	3	0
4	4	0

Out[93]:

	id	prediction
0	0	0
1	1	1
2	2	0
3	3	1
4	4	0
...	...	...
6507	6507	0
6508	6508	1
6509	6509	0
6510	6510	0
6511	6511	0

6512 rows × 2 columns

In [94]:

```
submission.to_csv('submission_5th_pycaret.csv', index=False)
```

- 실제 모델과 달리 평가 데이터가 공개되어 있음(Data Leakage). 이에 따라 평가가 가능.
- 모델링의 중요 지표의 일반화를 내려놓고, test데이터에 대한 overfitting하는 과정을 갖는다.

- Score: 0.87482

CatBoost 모델링

- Pycaret는 좋은 모델링 도구이지만 미세한 파라미터 조정이 어려운 부분이 있음.
- pycaret에서 가장 좋은 수치를 가진 catboost를 직접 호출하여 파라미터 조정을 거친다.(submission 제출)

In [96]:

```
raw_train = train.copy()
raw_test = test.copy()
```

In [97]:

```
from catboost import CatBoostClassifier
import catboost
print(catboost.__version__)
```

```
1.0.4
```

In [98]:

```
def preprocess(df):
    # null값이 count 출력
    print(df.apply(lambda x: sum(x.isnull()), axis=0))
    print(' ')

    # income column을 string에서 integer로 변경
    df['income_level'] = np.where(df.income == '<=50K', 0, 1)

    # 성별
    df['gender'] = df['sex'].map({'Male': 0, 'Female': 1}).astype(int)


    # 인종
    ethnicity_key = {'White': 0, 'Black': 1, 'Asian-Pac-Islander': 2,
                     'Amer-Indian-Eskimo': 3, 'Other': 4}

    df['ethnicity'] = df['race'].map(ethnicity_key).astype(int)

    # 국가
    origin_key = {'?': 0, 'United-States': 1, 'Mexico': 2, 'Philippines': 3,
                  'Germany': 4, 'Canada': 5, 'Puerto-Rico': 6, 'El-Salvador': 7,
                  'India': 8, 'Cuba': 9, 'England': 10, 'Jamaica': 11, 'South': 12,
                  'China': 13, 'Italy': 14, 'Dominican-Republic': 15, 'Vietnam': 16,
                  'Guatemala': 17, 'Japan': 18, 'Poland': 19, 'Columbia': 20, 'Taiwa
                  'Haiti': 22, 'Iran': 23, 'Portugal': 24, 'Nicaragua': 25, 'Peru':
                  'France': 27, 'Greece': 28, 'Ecuador': 29, 'Ireland': 30, 'Hong':
                  'Trinadad&Tobago': 32, 'Cambodia': 33, 'Laos': 34, 'Thailand': 35,
                  'Yugoslavia': 36, 'Outlying-US(Guam-USVI-etc)': 37, 'Hungary': 38,
                  'Honduras': 39, 'Scotland': 40, 'Holand-Netherlands': 41}

    df['native_country'] = df['native_country'].map(origin_key).astype(int)

    # 고용형태
    work_key = {'Private': 0, 'Self-emp-not-inc': 1, 'Local-gov': 2, '?': 3,
                'State-gov': 4, 'Self-emp-inc': 5, 'Federal-gov': 6,
                'Without-pay': 7, 'Never-worked': 8}

    df['work'] = df['workclass'].map(work_key).astype(int)

    # 결혼상태
    marital_status_key = {'Married-civ-spouse': 0, 'Never-married': 1, 'Divorced': 2
                          'Separated': 3, 'Widowed': 4, 'Married-spouse-absent': 5,
                          'Married-AF-spouse': 6}

    df['marital_status'] = df['marital_status'].map(marital_status_key).astype(int)

    # 업종
    occupation_key = {'Prof-specialty': 0, 'Craft-repair': 1, 'Exec-managerial': 2,
                      'Adm-clerical': 3, 'Sales': 4, 'Other-service': 5,
                      'Machine-op-inspct': 6, '?': 7, 'Transport-moving': 8,
                      'Handlers-cleaners': 9, 'Farming-fishing': 10, 'Tech-support':
                      'Protective-serv': 12, 'Priv-house-serv': 13, 'Armed-Forces':

    df['occupation'] = df['occupation'].map(occupation_key).astype(int)

    # 가족관계
    relationship_key = {'Husband': 0, 'Not-in-family': 1, 'Own-child': 2, 'Unmarried
                        'Wife': 4, 'Other-relative': 5}
```

```python
    df['relationship'] = df['relationship'].map(relationship_key).astype(int)

    # raw column 삭제
    df = df.drop(['income'], axis=1)
    df = df.drop(['sex'], axis=1)
    df = df.drop(['race'], axis=1)
    # df = df.drop(['native.country'], axis=1)
    df = df.drop(['workclass'], axis=1)
    # df = df.drop(['marital.status'], axis=1)
    df = df.drop(['education'], axis=1)
    # dummy = pd.get_dummies(df['education'], prefix='education')
    # del df['education']
    # df = pd.concat([df, dummy], axis=1)
    # df = df.drop(['education_num'], axis=1)

    # 주당 근무 시간
    df['hours_per_week'] = df['hours_per_week'].astype(int)
    # df.loc[df['hours_per_week'] < 40, 'hours_per_week'] = 0
    # df.loc[df['hours_per_week'] == 40, 'hours_per_week'] = 1
    # df.loc[df['hours_per_week'] > 40, 'hours_per_week'] = 2

    # 양도소득차
    df['capital_diff'] = df['capital_gain'] - df['capital_loss']
    #df['fnlwgt_log'] = np.log(df['fnlwgt'])
    #df['education_num'] /= 10
    # df['age_log'] = np.log(df['age'])
    #del df['fnlwgt']
    # del df['native_country']

    return df
```

In [99]:

```python
# 데이터 전처리
all_data = pd.concat([raw_train, raw_test])
all_data = preprocess(all_data)
train = all_data.iloc[:len(raw_train)]
test = all_data.iloc[len(raw_train):]

train_x = train.drop(['income_level'], axis=1)
train_y = train['income_level']

test_x = test.drop(['income_level'], axis=1)
```

```
age                  0
workclass            0
fnlwgt               0
education            0
education_num        0
marital_status       0
occupation           0
relationship         0
race                 0
sex                  0
capital_gain         0
capital_loss         0
hours_per_week       0
native_country       0
income            6512
dtype: int64
```

In [100]:

```
prediction = np.zeros(len(test_x))
learning_params = [
    {
    "learning_rate": 0.2,
    "iterations": 212,
    "depth": 4,
    "l2_leaf_reg": 3,
    "random_seed": 62,
    "random_strength": 1,
    "eval_metric": 'Accuracy'
    },
    {
    "learning_rate": 0.2,
    "iterations": 273,
    "depth": 4,
    "l2_leaf_reg": 3,
    "random_seed": 8,
    "random_strength": 1,
    "eval_metric": 'Accuracy'
    },
    {
    "learning_rate": 0.2,
    "iterations": 277,
    "depth": 4,
    "l2_leaf_reg": 3,
    "random_seed": 145,
    "random_strength": 1,
    "eval_metric": 'Accuracy'
    }]

for param in learning_params:
    model = CatBoostClassifier(**param)
    model.fit(train_x, train_y, verbose=False)
    prediction += model.predict(test_x)

prediction = prediction / 3
prediction[prediction < 1] = 0
prediction = prediction.astype(np.int64)
```

In [101]:

```
from sklearn.metrics import accuracy_score
```

In [102]:

```
# 학습셋의 성능
train_prediction = model.predict(train_x)
accuracy_score(train_y, train_prediction)
```

Out[102]:

```
0.8893623555606741
```

In [103]:

```
# 예측 점수가 낮은 결과들의 정확도
train_score_list = model.predict_proba(train_x)
low_score_indexes = np.where(np.logical_and(train_score_list[:,0] < 0.55,
                                            train_score_list[:,0] > 0.45))[0]
accuracy_score(train_y[low_score_indexes], train_prediction[low_score_indexes])
```

Out[103]:

```
0.5264116575591985
```

In [104]:

```
score_list = model.predict_proba(test_x)
```

- model의 예측치를 0.55 ~ 0.50, 0.45 ~ 0.50 2분류로 나눠서 임의로 지정하고 submission 제출을 통해서 best random값을 찾는다.

In [105]:

```
score_ranges = [
    (0.55, 0.50, 999, 30), # label 1 min
    (0.50, 0.45, 2000000, 9)  # label 0
]
for up, down, seed, random_range in score_ranges:
    indexes = np.where(np.logical_and(score_list[:,0] < up, score_list[:,0] > down))
    np.random.seed(seed)
    rand_val = np.random.randint(0,random_range,size=len(indexes))
    rand_threshold = np.random.randint(1, random_range -1)
    rand_val[rand_val<rand_threshold] = 0;
    rand_val[rand_val>=rand_threshold] = 1
    prediction[indexes] = rand_val
```

제출

In [106]:

```
submission = pd.read_csv(os.path.join(dirname, 'sample_submission.csv'))
submission['prediction'] = prediction
submission.to_csv('submission_6th_catBoost.csv', index=False)
```

- REF
  - https://velog.io/@jee-9/PyCaret-Tutorial-Docs%EC%99%80-%ED%95%A8%EA%BB%98-%EA%B0%84%EB%8B%A8%ED%95%98%EA%B2%8C-%EC%9D%B4%EC%9A%A9%ED%95%B4%EB%B3%B4%EA%B8%B0 (https://velog.io/@jee-9/PyCaret-Tutorial-Docs%EC%99%80-%ED%95%A8%EA%BB%98-%EA%B0%84%EB%8B%A8%ED%95%98%EA%B2%8C-%EC%9D%B4%EC%9A%A9%ED%95%B4%EB%B3%B4%EA%B8%B0)

- Score: 0.87040