Adult Income Prediction

If you want to be the first to be informed about new projects, please do not forget to follow us - by Fatma Nur AZMAN

Fatmanurazman.com | Linkedin | Github | Kaggle | Tableau

Understanding The Data

Project Description:

- **Adult Income Prediction** This dataset was obtained from UCI Machine Learning Repository. The aim of this problem is to classify adults in two different groups based on their income where group 1 has an income less than USD 50k and group 2 has an income of more than or equal to USD 50k. The data available at hand comes from Census 1994.

Domain Knowledge:

Economic Conditions

- Technological Revolution:

At the beginning of the 1990s, the widespread adoption of the internet and the rapid development of computer technology led to significant changes in the labor market. Information technology and service sectors grew rapidly, creating many new job opportunities.

- Economic Growth:

The US economy entered a significant growth period from the mid-1990s. This growth was supported by low inflation and low unemployment rates. However, economic opportunities were not equally distributed across all regions and groups.

Social and Political Situation

- Diversity and Immigration:

In the 1990s, the number of people immigrating to the US increased. Immigrants played a crucial role in the labor market and met labor demands in many sectors. This situation also led to some social tensions and debates.

- Education and Workforce:

The increasing importance of education levels in the labor market directly affected individuals' income levels. Higher-educated individuals generally worked in higher-paying jobs, while lower-educated individuals had to work in low-wage jobs.

Demographic Changes

- Aging Population:

The aging of the baby boomer generation began to put pressure on social security systems and healthcare services. The increasing number of individuals reaching retirement age also led to changes in the labor market.

- Women's Participation in the Workforce:

Women's participation in the workforce increased significantly in the 1990s. This led to an increase in household incomes and changes in gender roles in society.

Sectoral Changes

- Transformation of the Manufacturing Industry:

In the 1990s, while the manufacturing industry declined in some regions, the service and technology-based sectors grew. This transformation led to increased unemployment rates in some areas and economic imbalances.

- Globalization:

Globalization led to increased trade and investments. Many US companies moved their production facilities abroad while gaining access to global markets. This caused some uncertainties and changes in the labor market.

In this context, the data obtained from the 1994 Census reflects the aforementioned economic, social, and demographic changes. By examining the impact of education levels, gender, race, and occupations on income in the labor market, we can better understand the social dynamics of that period. These analyses can also contribute to understanding the changes and continuities in comparison with today's conditions.

About the Dataset

Dataset Descriptions:

- **Rows:** 32561
- **Columns:** 15

STT	Attribute Name	Unique Values
1	Age	Describes the age of individuals. Continuous.
2	Workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3	fnlwgt	Continuous. This is a weighting factor created by the US Census Bureau and indicates the number of people represented by each data entry.
4	education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
5	education-num	Number of years spent in education. Continuous.
6	marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
7	occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
8	relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
9	race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10	sex	Female, Male.
11	capital-gain	Represents the profit an individual makes from the sale of assets (e.g., stocks or real estate). Continuous.
12	capital-loss	Represents the loss an individual incurs from the sale of assets (e.g., stocks or real estate). Continuous.
13	hours-per-week	Continuous.
14	native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad & Tobago, Peru, Hong, Netherlands.
15	salary	>50K, <=50K.

Table of CONTENTS

Import Libraries and Data Review

In [58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

%matplotlib inline

from sklearn.impute import SimpleImputer

from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV, cross_valio
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.compose import make_column_transformer


from sklearn.metrics import make_scorer, precision_score, recall_score, f1_scor
from sklearn.metrics import PrecisionRecallDisplay, roc_curve, average_precisic
from sklearn.metrics import RocCurveDisplay, roc_auc_score, auc
from sklearn.metrics import confusion_matrix, classification_report, Confusionl


from yellowbrick.regressor import ResidualsPlot, PredictionError

import warnings
warnings.filterwarnings("ignore")
```

In [59]:
```
df0 = pd.read_csv('adult.csv')
df = df0.copy()
```

In [3]:
```
df.shape
```

Out[3]: (32561, 15)

In [4]:
```
df.head()
```

Out[4]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	rela
0	90	?	77053	HS-grad	9	Widowed	?	
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	
2	66	?	186061	Some-college	10	Widowed	?	U
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	U
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	C

In [6]:
```
df.tail()
```

Out[6]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical

Exploratory Data Analysis (EDA)

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [60]:
```
df[df == '?'] = np.nan
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      30718 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  31978 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [9]:
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%
age	32561.0	38.581647	13.640433	17.0	28.0	37.0
fnlwgt	32561.0	189778.366512	105549.977697	12285.0	117827.0	178356.0
education.num	32561.0	10.080679	2.572720	1.0	9.0	10.0
capital.gain	32561.0	1077.648844	7385.292085	0.0	0.0	0.0
capital.loss	32561.0	87.303830	402.960219	0.0	0.0	0.0
hours.per.week	32561.0	40.437456	12.347429	1.0	40.0	40.0

In [10]:
```
df.describe(include="object").T
```

Out[10]:

	count	unique	top	freq
workclass	30725	8	Private	22696
education	32561	16	HS-grad	10501
marital.status	32561	7	Married-civ-spouse	14976
occupation	30718	14	Prof-specialty	4140
relationship	32561	6	Husband	13193
race	32561	5	White	27816
sex	32561	2	Male	21790
native.country	31978	41	United-States	29170
income	32561	2	<=50K	24720

In [6]:
```
df.duplicated().sum()
```

Out[6]: 24

In [61]:
```
def duplicate_values(df):
    print("Duplicate check...")
    num_duplicates = df.duplicated(subset=None, keep='first').sum()
    if num_duplicates > 0:
        print("There are", num_duplicates, "duplicated observations in the data
        df.drop_duplicates(keep='first', inplace=True)
        print(num_duplicates, "duplicates were dropped!")
        print("No more duplicate rows!")
    else:
        print("There are no duplicated observations in the dataset.")
```

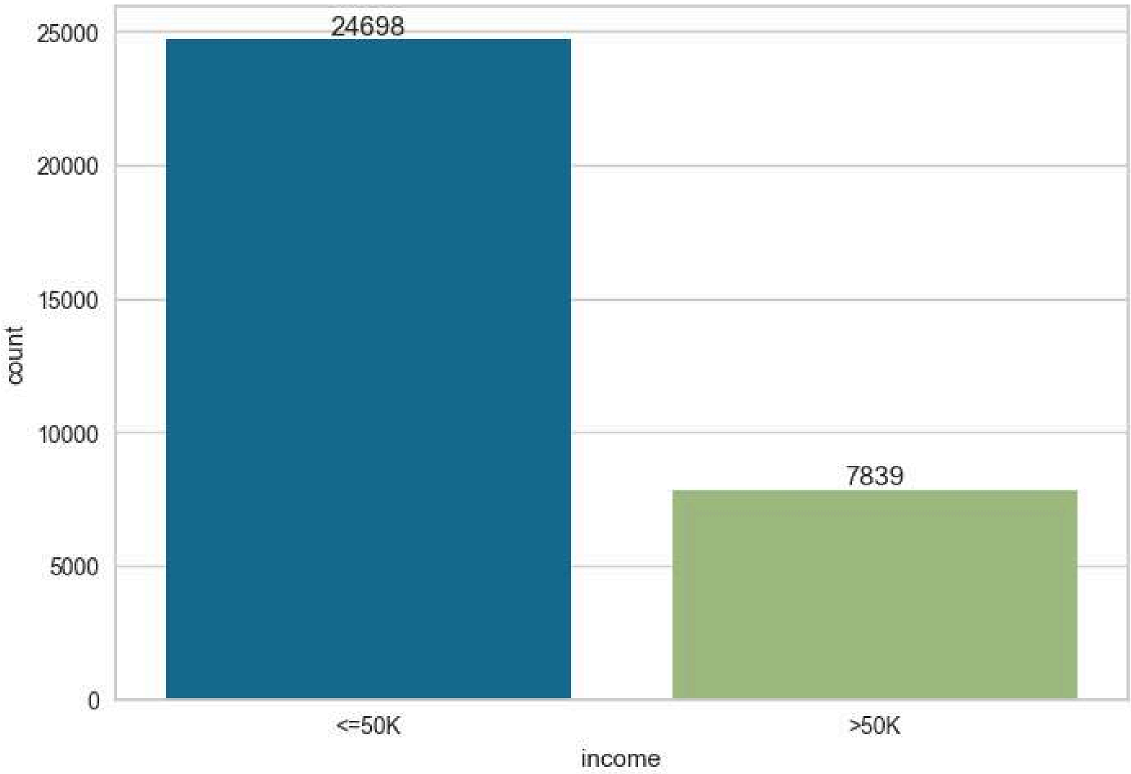
In [62]:
```
duplicate_values(df)
```

```
Duplicate check...
There are 24 duplicated observations in the dataset.
24 duplicates were dropped!
No more duplicate rows!
```

In [9]:
```
df.isnull().sum().sum()
```

Out[9]: 4261

In [15]:
```
ax = sns.countplot(x="income", data=df)
ax.bar_label(ax.containers[0]);
```



Our data is a unbalance data.

Features Summary

In [15]:
```
# !pip install ipywidgets ydata-profiling
#from ydata_profiling import ProfileReport
#profile = ProfileReport(df, title="Profiling Report")
#profile.to_file("profiling_report.html")
```

In [16]:
```
#!pip install summarytools
from summarytools import dfSummary
dfSummary(df)
```

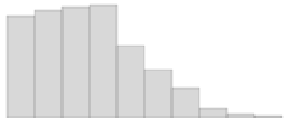
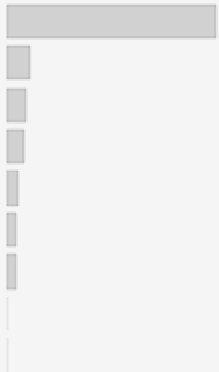
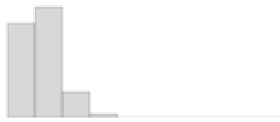
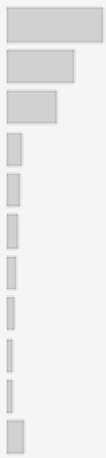
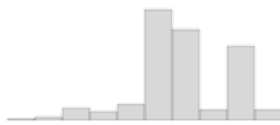
Out[16]:

Data Frame Summary

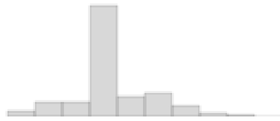
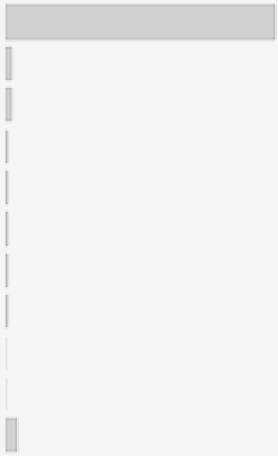
df

Dimensions: 32,537 x 15

Duplicates: 0

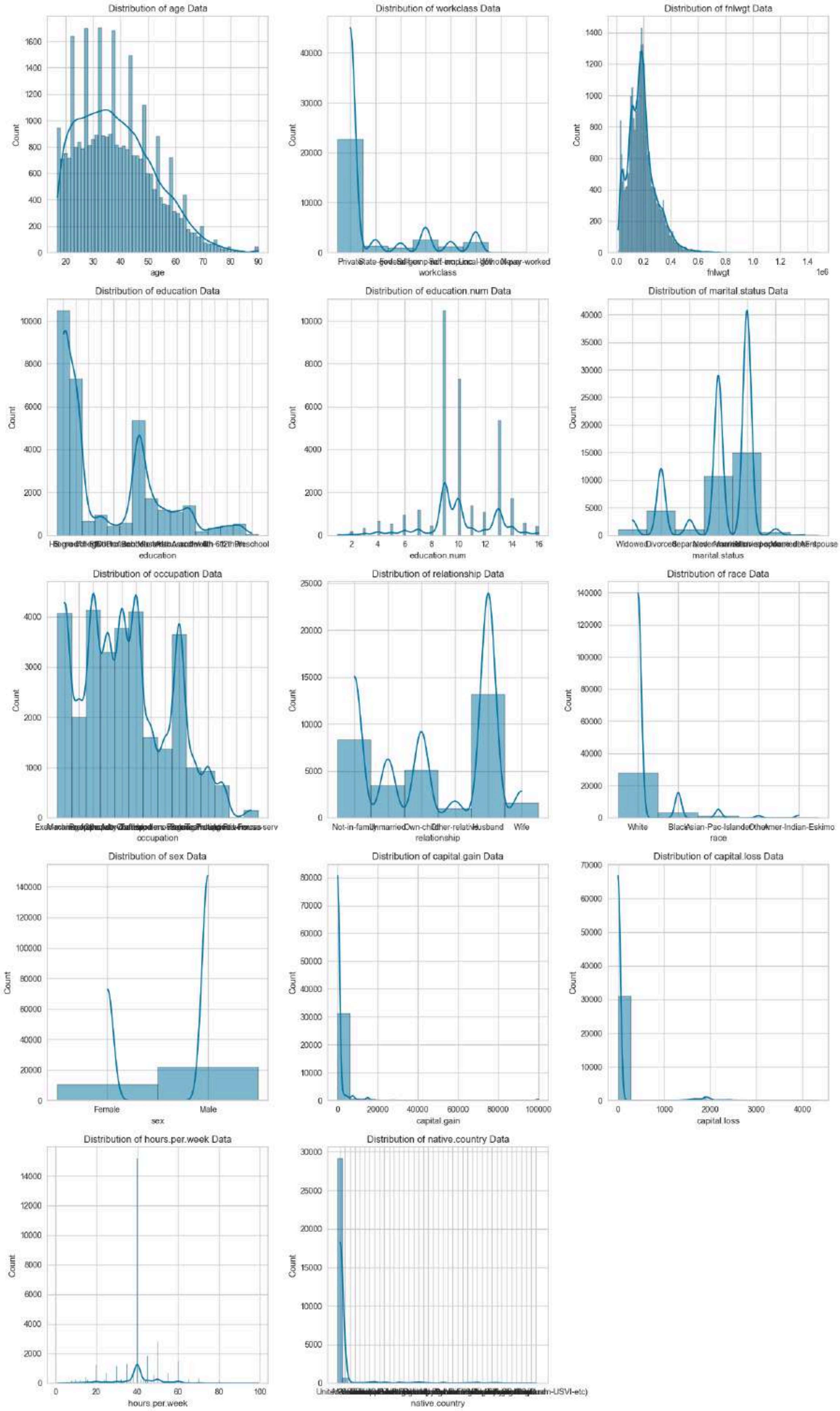
No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missi
1	**age** [int64]	Mean (sd) : 38.6 (13.6) min < med < max: 17.0 < 37.0 < 90.0 IQR (CV) : 20.0 (2.8)	73 distinct values		0 (0.0%)
2	**workclass** [object]	1. Private 2. Self-emp-not-inc 3. Local-gov 4. nan 5. State-gov 6. Self-emp-inc 7. Federal-gov 8. Without-pay 9. Never-worked	22,673 (69.7%) 2,540 (7.8%) 2,093 (6.4%) 1,836 (5.6%) 1,298 (4.0%) 1,116 (3.4%) 960 (3.0%) 14 (0.0%) 7 (0.0%)		1,836 (5.6%)
3	**fnlwgt** [int64]	Mean (sd) : 189780.8 (105556.5) min < med < max: 12285.0 < 178356.0 < 1484705.0 IQR (CV) : 119166.0 (1.8)	21,648 distinct values		0 (0.0%)
4	**education** [object]	1. HS-grad 2. Some-college 3. Bachelors 4. Masters 5. Assoc-voc 6. 11th 7. Assoc-acdm 8. 10th 9. 7th-8th 10. Prof-school 11. other	10,494 (32.3%) 7,282 (22.4%) 5,353 (16.5%) 1,722 (5.3%) 1,382 (4.2%) 1,175 (3.6%) 1,067 (3.3%) 933 (2.9%) 645 (2.0%) 576 (1.8%) 1,908 (5.9%)		0 (0.0%)
5	**education.num** [int64]	Mean (sd) : 10.1 (2.6) min < med < max: 1.0 < 10.0 < 16.0 IQR (CV) : 3.0 (3.9)	16 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missi
6	**marital.status** [object]	1. Married-civ-spouse 2. Never-married 3. Divorced 4. Separated 5. Widowed 6. Married-spouse-absent 7. Married-AF-spouse	14,970 (46.0%) 10,667 (32.8%) 4,441 (13.6%) 1,025 (3.2%) 993 (3.1%) 418 (1.3%) 23 (0.1%)		0 (0.0%)
7	**occupation** [object]	1. Prof-specialty 2. Craft-repair 3. Exec-managerial 4. Adm-clerical 5. Sales 6. Other-service 7. Machine-op-inspct 8. nan 9. Transport-moving 10. Handlers-cleaners 11. other	4,136 (12.7%) 4,094 (12.6%) 4,065 (12.5%) 3,768 (11.6%) 3,650 (11.2%) 3,291 (10.1%) 2,000 (6.1%) 1,843 (5.7%) 1,597 (4.9%) 1,369 (4.2%) 2,724 (8.4%)		1,843 (5.7%)
8	**relationship** [object]	1. Husband 2. Not-in-family 3. Own-child 4. Unmarried 5. Wife 6. Other-relative	13,187 (40.5%) 8,292 (25.5%) 5,064 (15.6%) 3,445 (10.6%) 1,568 (4.8%) 981 (3.0%)		0 (0.0%)
9	**race** [object]	1. White 2. Black 3. Asian-Pac-Islander 4. Amer-Indian-Eskimo 5. Other	27,795 (85.4%) 3,122 (9.6%) 1,038 (3.2%) 311 (1.0%) 271 (0.8%)		0 (0.0%)
10	**sex** [object]	1. Male 2. Female	21,775 (66.9%) 10,762 (33.1%)		0 (0.0%)
11	**capital.gain** [int64]	Mean (sd) : 1078.4 (7388.0) min < med < max: 0.0 < 0.0 < 99999.0 IQR (CV) : 0.0 (0.1)	119 distinct values		0 (0.0%)
12	**capital.loss** [int64]	Mean (sd) : 87.4 (403.1) min < med < max: 0.0 < 0.0 < 4356.0 IQR (CV) : 0.0 (0.2)	92 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missi
13	**hours.per.week** [int64]	Mean (sd) : 40.4 (12.3) min < med < max: 1.0 < 40.0 < 99.0 IQR (CV) : 5.0 (3.3)	94 distinct values		0 (0.0%)
14	**native.country** [object]	1. United-States 2. Mexico 3. nan 4. Philippines 5. Germany 6. Canada 7. Puerto-Rico 8. El-Salvador 9. India 10. Cuba 11. other	29,153 (89.6%) 639 (2.0%) 582 (1.8%) 198 (0.6%) 137 (0.4%) 121 (0.4%) 114 (0.4%) 106 (0.3%) 100 (0.3%) 95 (0.3%) 1,292 (4.0%)		582 (1.8%)
15	**income** [object]	1. <=50K 2. >50K	24,698 (75.9%) 7,839 (24.1%)		0 (0.0%)

In [17]:

```
import math
num_cols = df.iloc[:, :-1].shape[1]
num_rows = math.ceil(num_cols / 3)

plt.figure(figsize=(15, 5 * num_rows))
for i, col in enumerate(df.iloc[:, :-1].columns, 1):
    plt.subplot(num_rows, 3, i)
    plt.title(f"Distribution of {col} Data")
    sns.histplot(df[col], kde=True)
    plt.tight_layout()
plt.show()
```



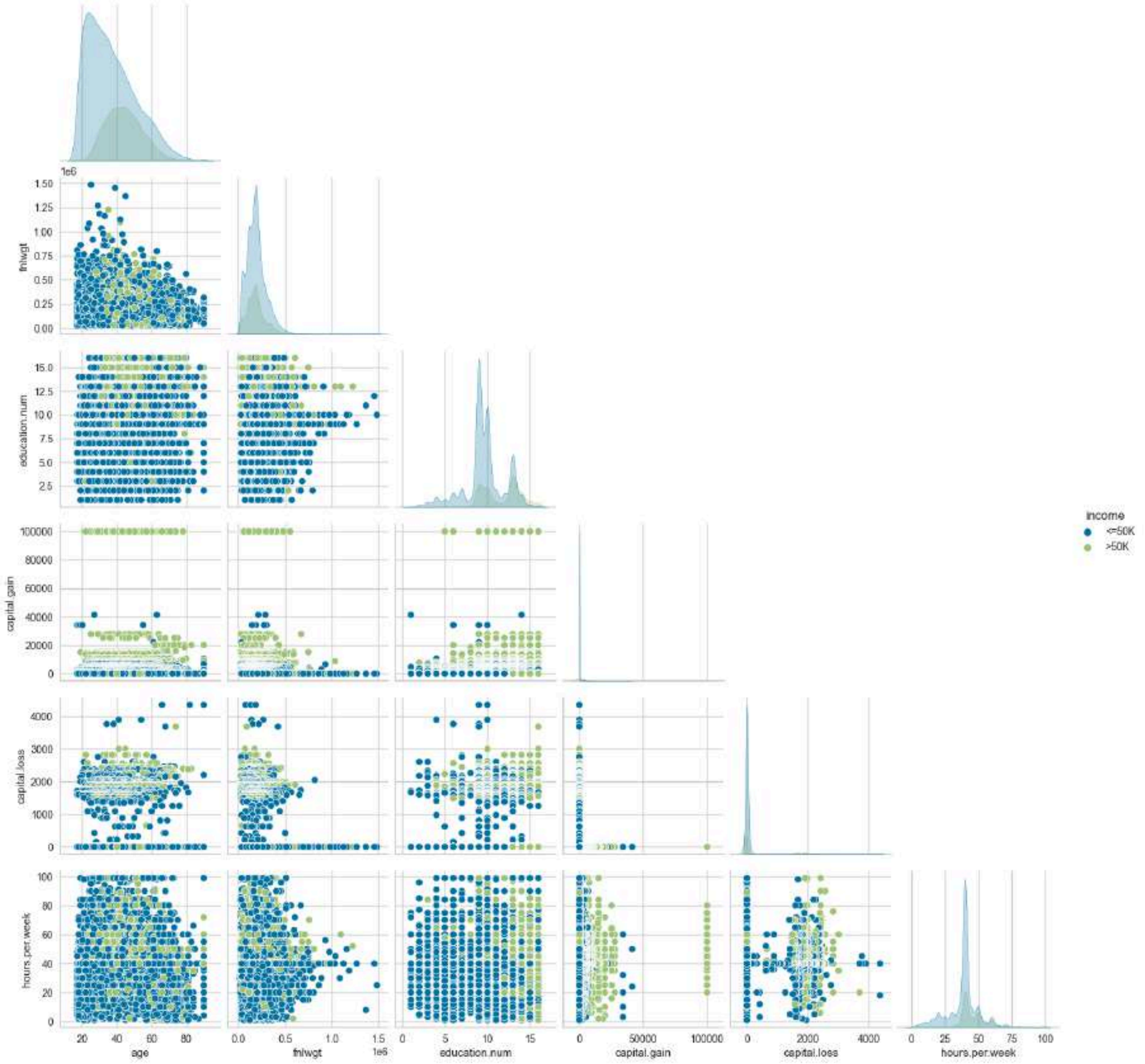
In [10]:
```
num_cols= df.select_dtypes('number').columns
```

```
skew_limit = 0.75            # define a limit above which we will log trans
skew_vals = df[num_cols].skew()


# Showing the skewed columns
skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))
skew_cols
```

Out[10]:

	Skew
capital.gain	11.949403
capital.loss	4.592702
fnlwgt	1.447703

In [20]:
```
sns.pairplot(df, hue= "income", corner=True);
```



In [63]:
```
cat_features = df.select_dtypes(include=['object']).columns.tolist()
cat_features = [col for col in cat_features if col != 'income']
cat_features
```

Out[63]:
```
['workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country']
```

In [64]:
```
num_features = df.select_dtypes(include=['number']).columns.tolist()
```

In [65]:
```
df['income'] = df['income'].apply(lambda x: 0 if x == '<=50K' else 1)
```

Handling Missing Values

In [24]:
```
df.isnull().sum().sum()
```

Out[24]:
```
4261
```

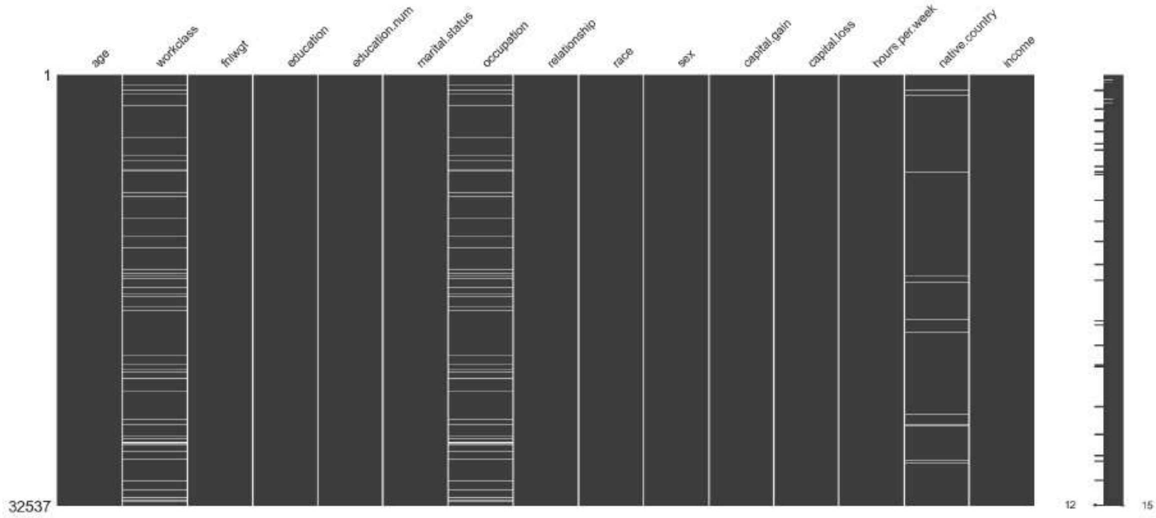
In [25]:
```
missing_count = df.isnull().sum()
value_count = df.isnull().count()
missing_percentage = round(missing_count / value_count * 100, 2)
missing_df = pd.DataFrame({"count": missing_count, "percentage": missing_percer
missing_df
```

Out[25]:

	count	percentage
age	0	0.00
workclass	1836	5.64
fnlwgt	0	0.00
education	0	0.00
education.num	0	0.00
marital.status	0	0.00
occupation	1843	5.66
relationship	0	0.00
race	0	0.00
sex	0	0.00
capital.gain	0	0.00
capital.loss	0	0.00
hours.per.week	0	0.00
native.country	582	1.79
income	0	0.00

In [30]:
```
# !pip install missingno
import missingno as msno
```

```
msno.matrix(df);
```



In [66]:
```
num_imputer = SimpleImputer(strategy='median')
cat_imputer = SimpleImputer(strategy='most_frequent')

# Impute numerical columns
df[num_features] = num_imputer.fit_transform(df[num_features])

# Impute categorical columns
df[cat_features] = cat_imputer.fit_transform(df[cat_features])
```

In [27]:
```
# Let's observe our data in a table

def get_unique_values(df):

    output_data = []

    for col in df.columns:

        # If the number of unique values in the column is less than or equal to
        if df.loc[:, col].nunique() <= 10:
            # Get the unique values in the column
            unique_values = df.loc[:, col].unique()
            # Append the column name, number of unique values, unique values, 
            output_data.append([col, df.loc[:, col].nunique(), unique_values, 
        else:
            # Otherwise, append only the column name, number of unique values,
            output_data.append([col, df.loc[:, col].nunique(),"-", df.loc[:, c

    output_df = pd.DataFrame(output_data, columns=['Column Name', 'Number of Un

    return output_df
```

In [28]:
```
get_unique_values(df)
```

Out[28]:

	Column Name	Number of Unique Values	Unique Values	Data Type
0	age	73	-	float64
1	workclass	8	[Private, State-gov, Federal-gov, Self-emp-not...	object
2	fnlwgt	21648	-	float64
3	education	16	-	object
4	education.num	16	-	float64
5	marital.status	7	[Widowed, Divorced, Separated, Never-married, ...	object
6	occupation	14	-	object
7	relationship	6	[Not-in-family, Unmarried, Own-child, Other-re...	object
8	race	5	[White, Black, Asian-Pac-Islander, Other, Amer...	object
9	sex	2	[Female, Male]	object
10	capital.gain	119	-	float64
11	capital.loss	92	-	float64
12	hours.per.week	94	-	float64
13	native.country	41	-	object
14	income	2	[0, 1]	int64

In [67]:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2,
                    subplot_titles=("Unique values per Categorical feature", "U

for col_type, col, color in [("exclude", 1, '#016CC9'), ("include", 2, '#DEB078
    temp_data = df.select_dtypes(**{col_type: "number"}).nunique().sort_values(
    fig.add_trace(go.Bar(x=temp_data.index, y=temp_data.values, marker=dict(col

fig.show()
```

Feature Engineering and Outliers

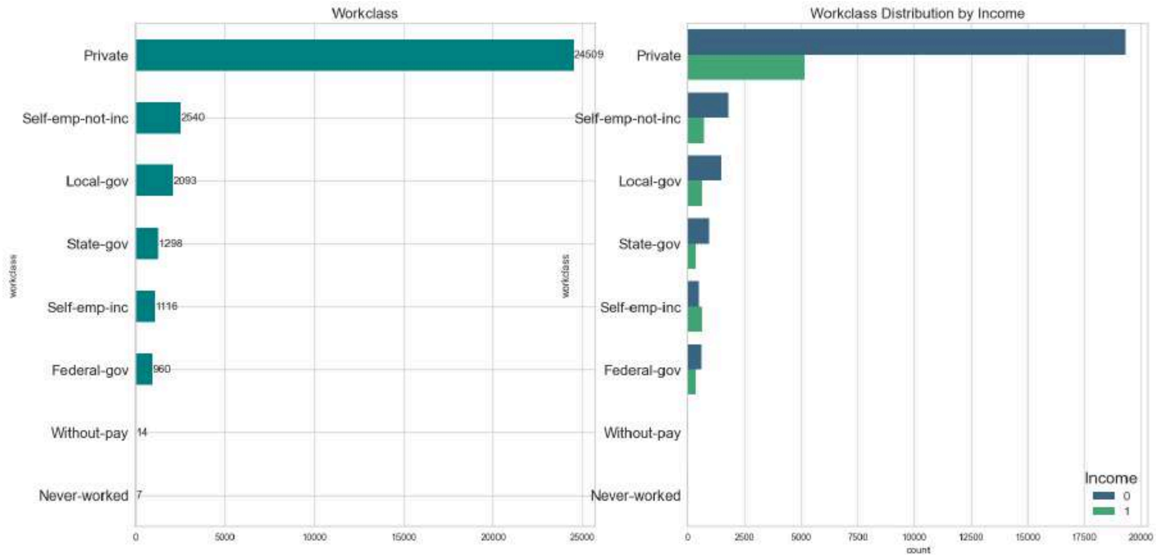
Categorical Features

In [11]:
```
df[cat_features].columns
```

Out[11]:
```
Index(['workclass', 'education', 'marital.status', 'occupation',
       'relationship', 'race', 'sex', 'native.country'],
      dtype='object')
```

In [31]:
```
sorted_workclass = ['Private', 'Self-emp-not-inc', 'Local-gov', 'State-gov', 'S

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

counts = df['workclass'].value_counts().reindex(sorted_workclass[::-1])
counts.plot(kind="barh", ax=ax1, color="teal")
ax1.set_title('Workclass', fontsize=16)
ax1.bar_label(ax1.containers[0], labels=counts.values, fontsize=12)
ax1.tick_params(axis='y', labelsize=16)

# 2.grafig: Workclass Distribution by Income
sns.countplot(y=df["workclass"], hue=df['income'].astype(str), ax=ax2, palette=
ax2.set_title('Workclass Distribution by Income', fontsize=16)
```

```
ax2.legend(title='Income', loc='lower right', fontsize=12, title_fontsize='18')
ax2.tick_params(axis='y', labelsize=16);
```

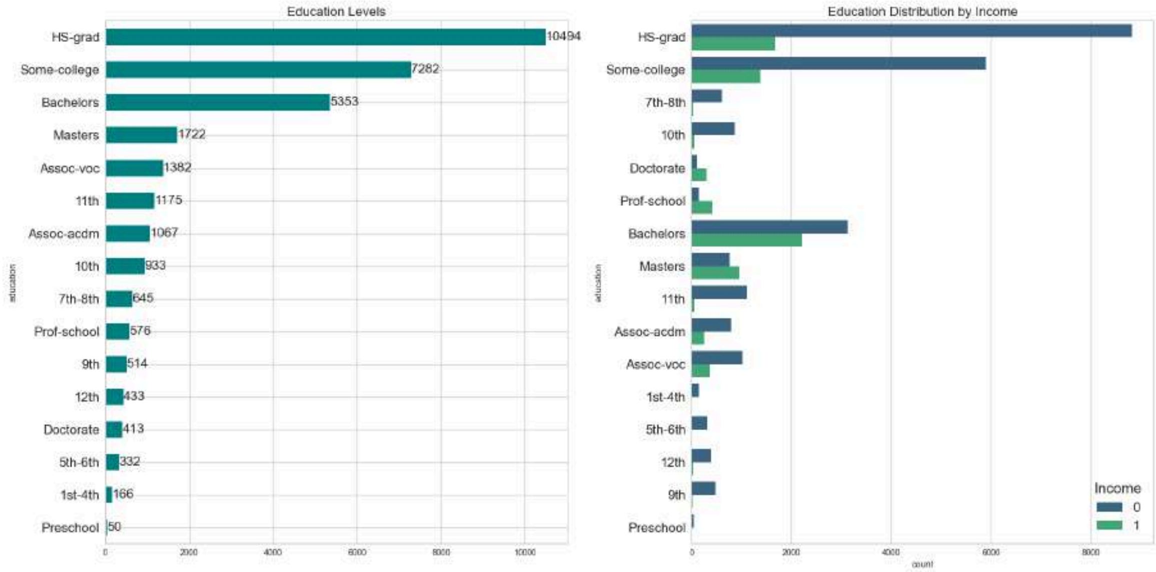


General Insights

- The private sector is the most dominant category among the work classes and creates a significant disparity in income distribution.
- Among self-employed individuals, those who are incorporated earn higher incomes compared to those who are not incorporated.
- For local, state, and federal government jobs, the low-income category is dominant; however, a significant portion also falls into the high-income category.
- Individuals who work without pay and those who have never worked are generally found in the low-income category.

In [32]:

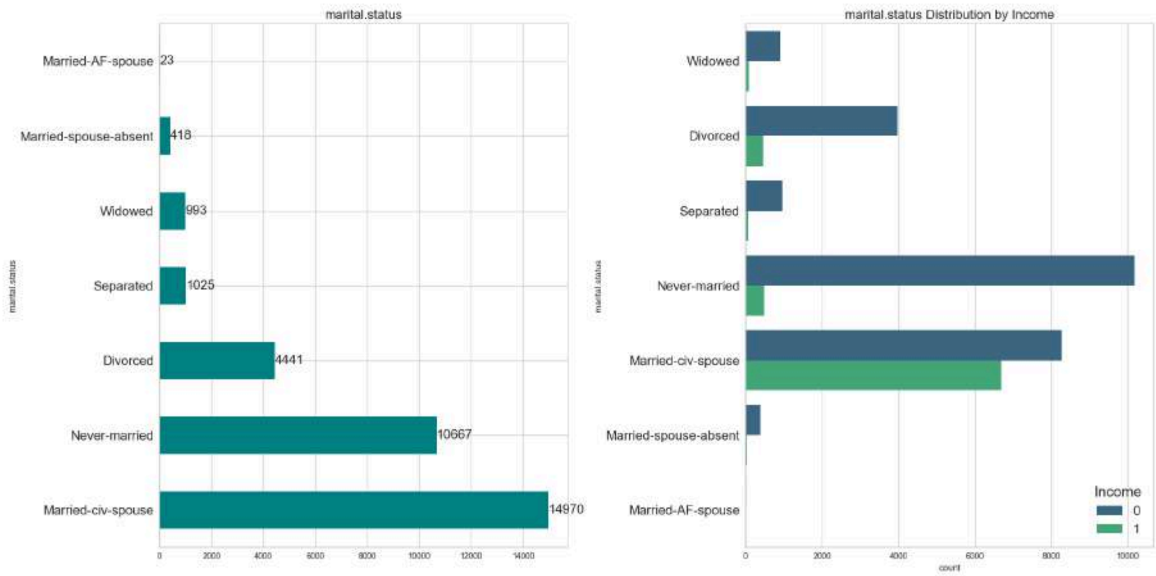
```
sorted_education = df['education'].value_counts().index[::-1]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Birinci grafik: Top Education Levels
counts = df['education'].value_counts().reindex(sorted_education)
counts.plot(kind="barh", ax=ax1, color="teal")
ax1.set_title('Education Levels', fontsize=16)
ax1.bar_label(ax1.containers[0], labels=counts.values, fontsize=16)
ax1.tick_params(axis='y', labelsize=16)

# İkinci grafik: Education Distribution by Income
sns.countplot(y=df["education"], hue=df['income'].astype(str), ax=ax2, palette=
ax2.set_title('Education Distribution by Income', fontsize=16)
ax2.legend(title='Income', loc='lower right', fontsize=16, title_fontsize='18')
ax2.tick_params(axis='y', labelsize=16)

plt.tight_layout()
plt.show()
```



In [68]:
```
df['education'].replace(['1st-4th', '5th-6th'], 'elementary_school', inplace=Tr
df['education'].replace(['7th-8th', '9th', '10th', '11th', '12th'], 'secondary_
df['education'].replace(['Assoc-acdm', 'Assoc-voc'], 'Assoc', inplace=True)
```

Category Merging: Dividing education levels into too many categories can complicate data analysis and modeling processes. Therefore, similar levels have been combined to form larger and more meaningful categories.

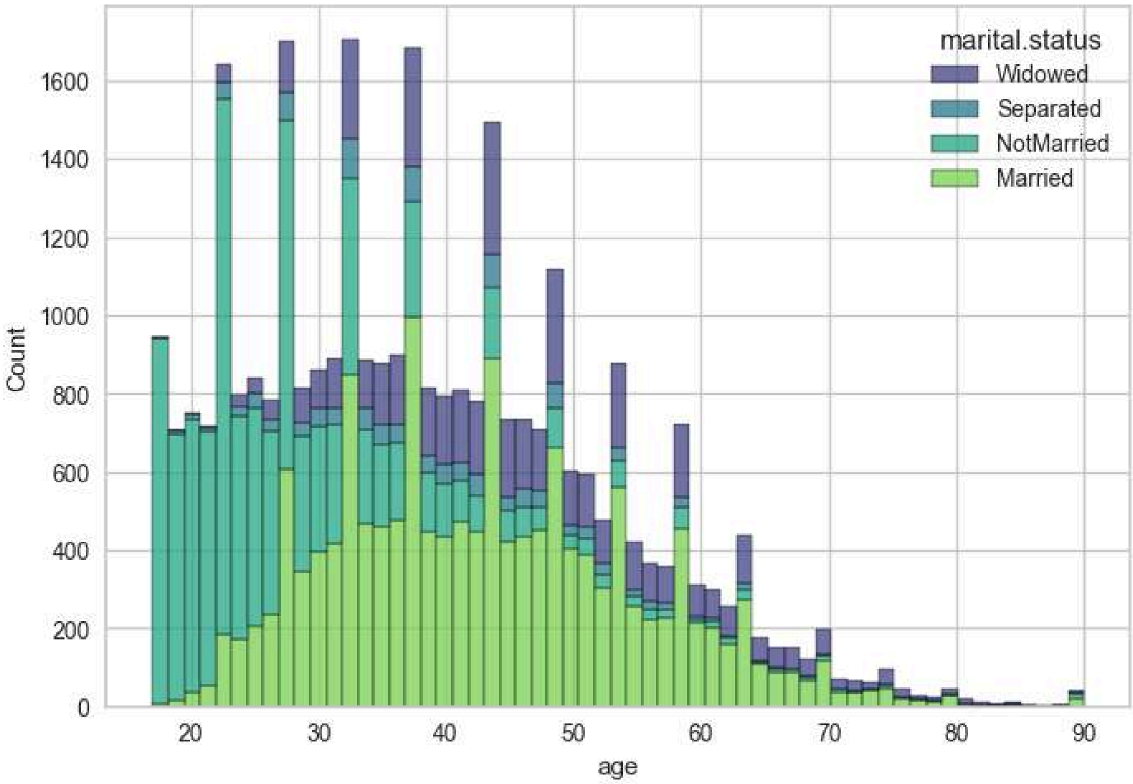
In [34]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Birinci grafik: Top Education Levels
counts = df['marital.status'].value_counts()
counts.plot(kind="barh", ax=ax1, color="teal")
ax1.set_title('marital.status', fontsize=16)
ax1.bar_label(ax1.containers[0], labels=counts.values, fontsize=16)
ax1.tick_params(axis='y', labelsize=16)

# İkinci grafik: Education Distribution by Income
sns.countplot(y=df["marital.status"], hue=df['income'].astype(str), ax=ax2, pal
ax2.set_title('marital.status Distribution by Income', fontsize=16)
ax2.legend(title='Income', loc='lower right', fontsize=16, title_fontsize='18')
ax2.tick_params(axis='y', labelsize=16)

plt.tight_layout()
plt.show()
```



In [69]:
```
df['marital.status'].replace(
    ['Never-married'], 'NotMarried', inplace=True
)
df['marital.status'].replace(
    ['Married-AF-spouse', 'Married-civ-spouse'], 'Married', inplace=True
)
df['marital.status'].replace(
    ['Married-spouse-absent', 'Separated'], 'Separated', inplace=True
)
df['marital.status'].replace(
    ['Divorced', 'Widowed'], 'Widowed', inplace=True
)
```

Marital Status Categories Merging In order to simplify the analysis and improve model performance, we combined similar marital status categories. This helps in reducing the number of distinct categories, making the data more manageable and the results more interpretable.

In [41]:
```
sns.histplot(data=df, x='age', hue='marital.status', multiple='stack', palette=
```

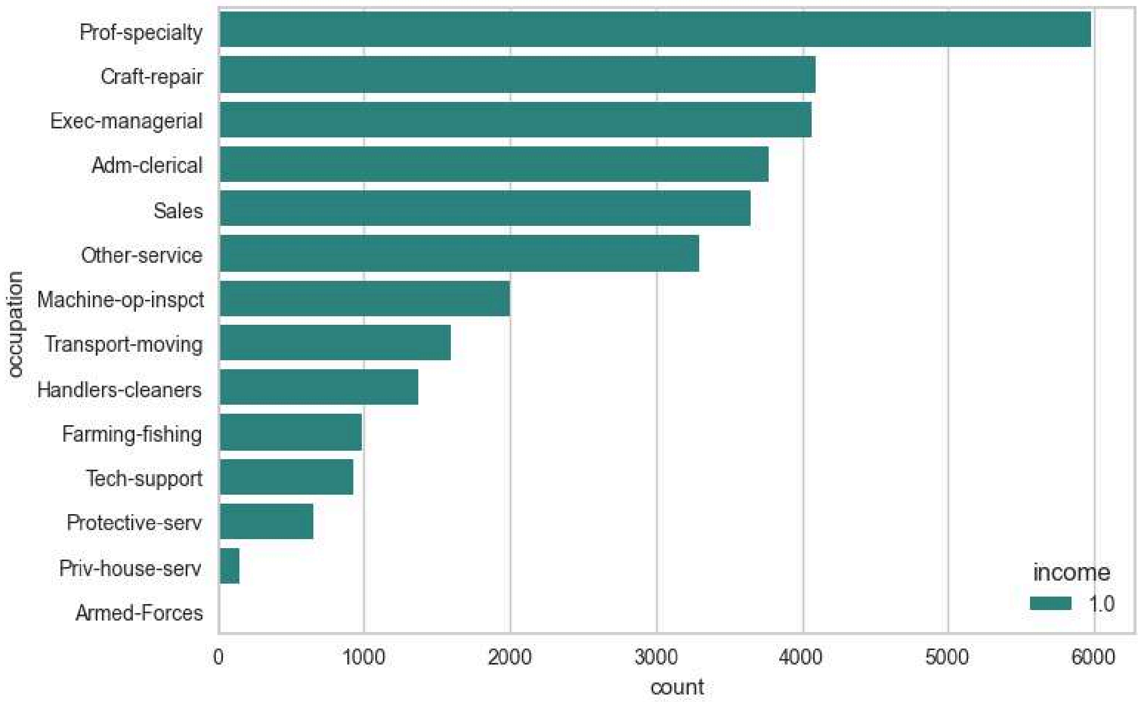


General Insights

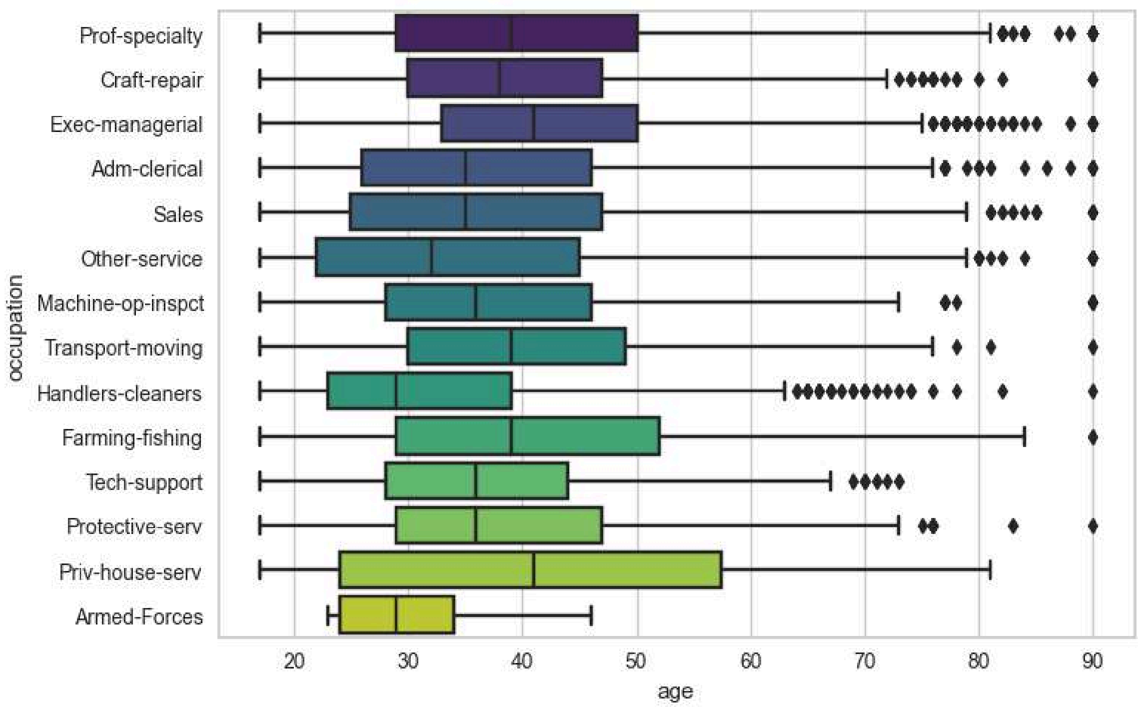
Marriage and Age: Marriage rates are low among young adults, peak in middle age, and decline again in older age. This indicates that focusing on education and career is common in early life, marriage and family building are more prevalent in middle age, and loss of a spouse increases in older age.

- **Tendency Not to Marry**: The non-marriage rates are higher among younger age groups, suggesting that education and career-oriented lifestyles are more common in modern societies.
- **Loss of Spouse and Separations**: Widowhood is more common in older age, while separations are more concentrated in middle age. This suggests that both increased rates of spouse loss due to health reasons and midlife crises or marital problems are more frequent in these age groups.

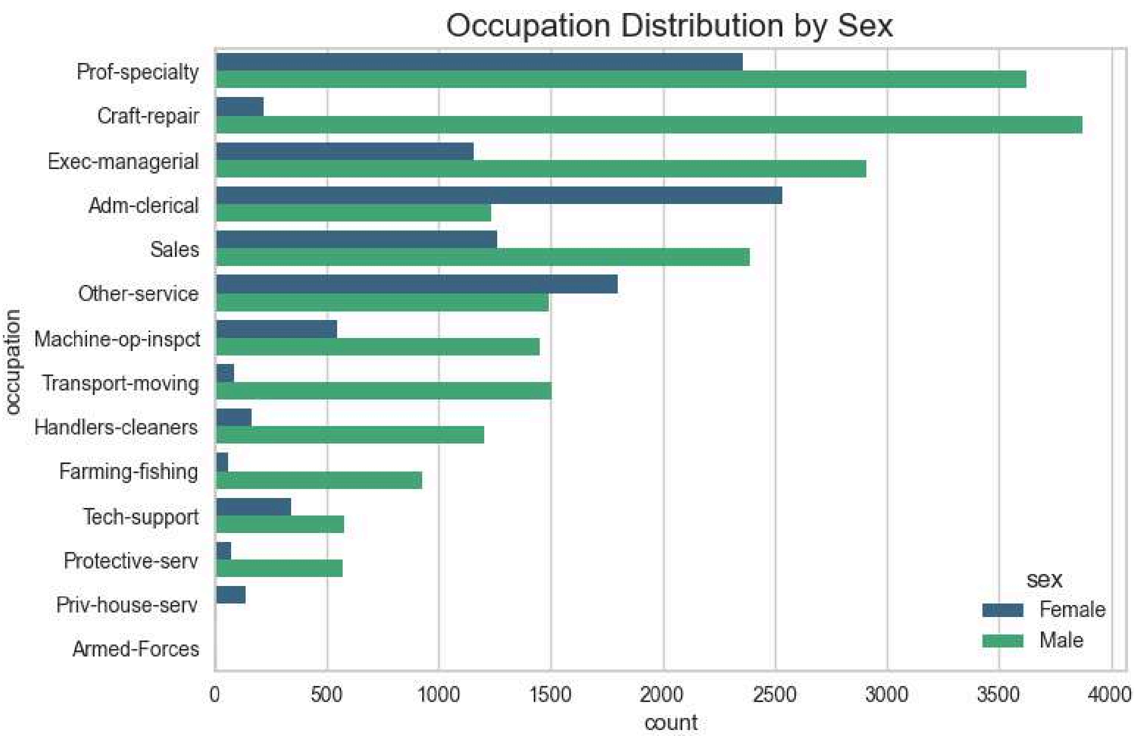
In [42]:
```
sns.countplot(y='occupation', hue='income', data=df, order=df['occupation'].va
```



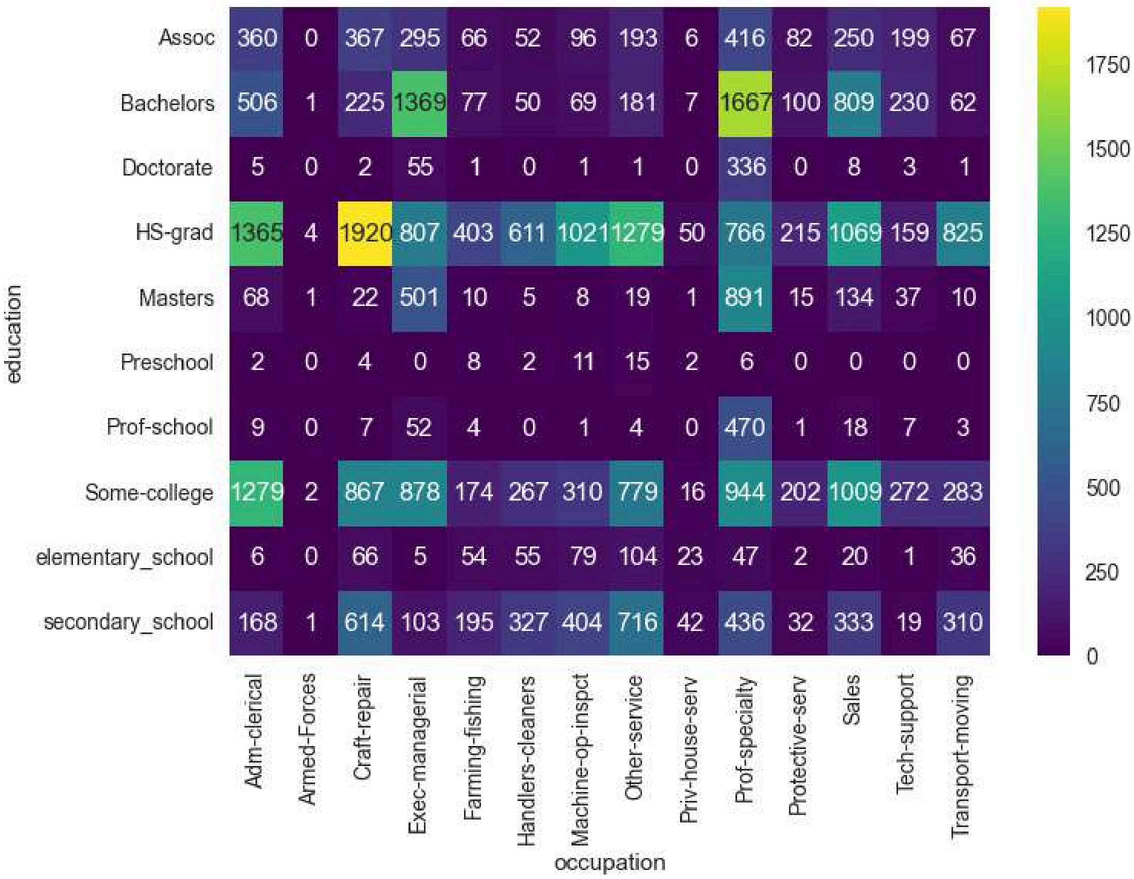
In [43]:
```
sns.boxplot(y='occupation', x='age', data=df, order=df['occupation'].value_cou
```



In [44]:
```
sns.countplot(y='occupation', hue='sex', data=df, order=df['occupation'].value_
plt.title('Occupation Distribution by Sex', fontsize=16);
```



In [46]:
```
pivot_table = df.pivot_table(index='education', columns='occupation', aggfunc=
sns.heatmap(pivot_table, annot=True, fmt='d', cmap='viridis');
```

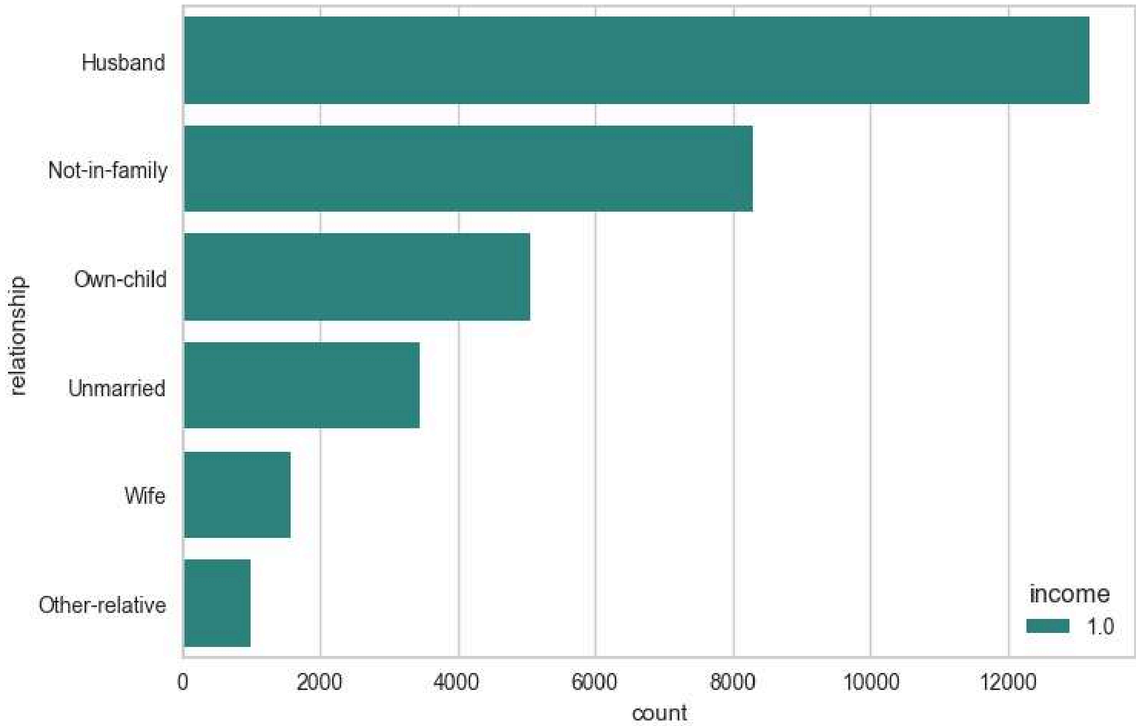


General Insights

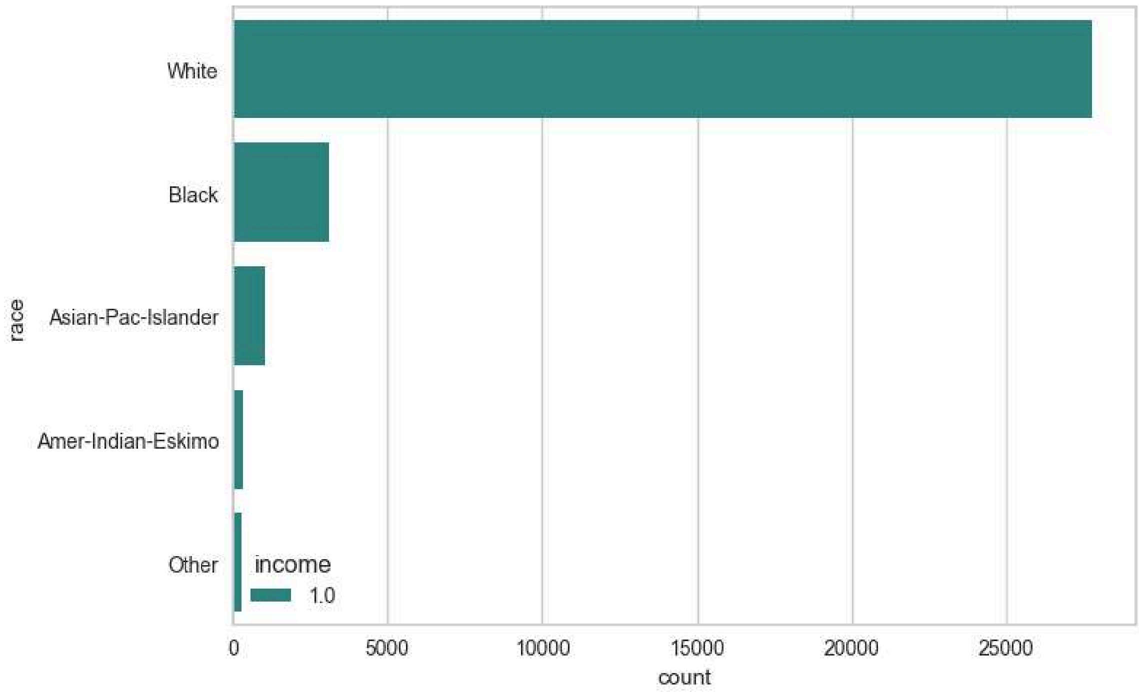
- **General InsightsIncome and Occupation**: Professional and managerial roles yield higher incomes, while service and manual labor roles are lower-income.

- **Age and Occupation**: Older individuals are more prevalent in high-responsibility roles, whereas younger individuals occupy more entry-level or physically demanding jobs.
- **Gender and Occupation**: There are significant gender disparities, with males dominating technical and managerial fields and females more present in clerical and service roles.
- **Education and Occupation**: Higher education levels correlate with higher-level occupations, whereas lower education levels are sufficient for service and manual jobs.

In [47]:
```
sns.countplot(y='relationship', hue='income', data=df, order=df['relationship']
```

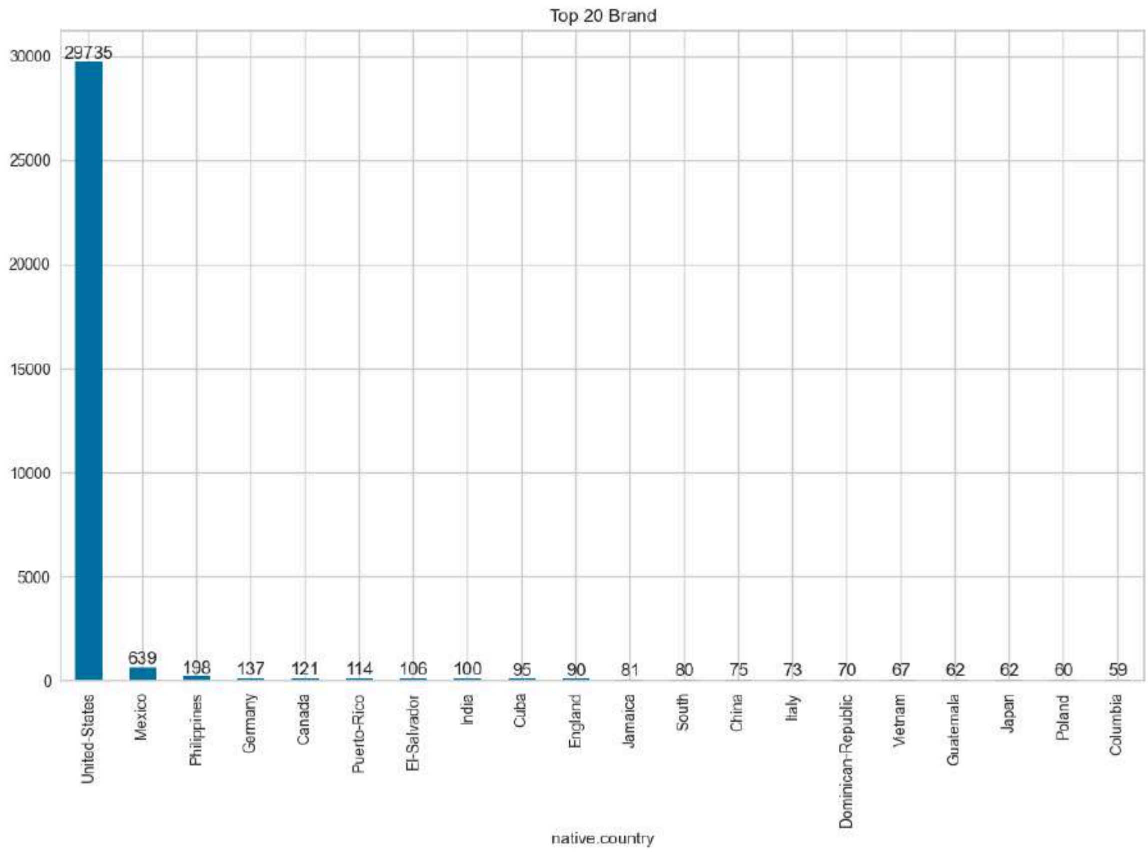


In [48]:
```
sns.countplot(y='race', hue='income', data=df, order=df['race'].value_counts()
```



In [71]:
```
df['race'].replace(['Asian-Pac-Islander', 'Amer-Indian-Eskimo', 'Other'],' Othe
```

In [15]:
```
fig = plt.figure(figsize = (10,6))
ax = fig.add_axes([0,0,1,1])
counts = df["native.country"].value_counts().sort_values(ascending=False).head(
counts.plot(kind = "bar")
plt.title('Top 20 Brand')
plt.xlabel('native.country')
plt.xticks(rotation = 90)
ax.bar_label(ax.containers[0], labels=counts.values, fontsize=12);
```



In [70]:
```
df['native.country'] = df['native.country'].replace({
    "USA": "United-States"
}).apply(lambda x: "United-States" if x == "United-States" else ("Mexico" if x
```

General Insights:

- **Data Imbalance:**

The data is heavily skewed towards individuals from the United States, which could impact the generalizability of any models or analyses performed.

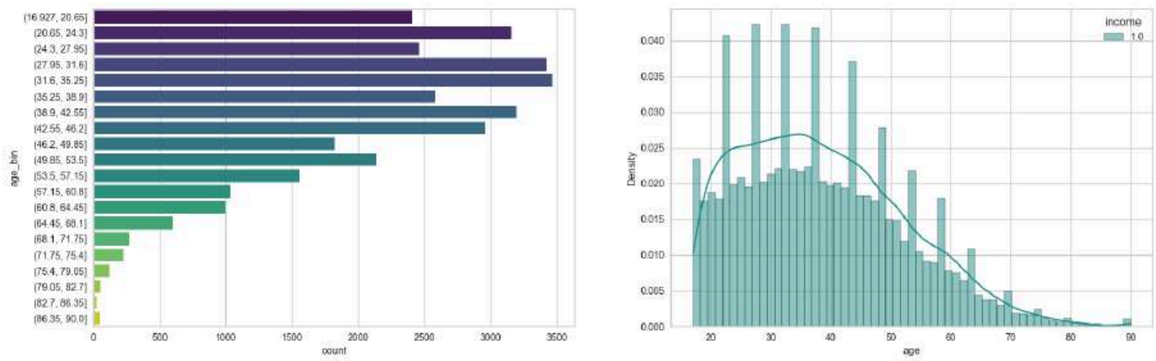
The dataset is predominantly composed of individuals from the United States, with a minor but noticeable representation from Mexico and a variety of other countries. This heavy imbalance towards the US population suggests the need for careful handling of data to avoid biases.

Given the significant representation from Mexico, segmented analyses (e.g., comparing outcomes between US natives and Mexican immigrants) might be feasible and insightful.

For other countries with smaller representations, aggregated analyses might be more appropriate.

Numerical Features

In [54]:
```
df['age_bin'] = pd.cut(df['age'], bins=20)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))
sns.countplot(y='age_bin', data=df, palette='viridis', ax=ax1)
sns.histplot(data=df, x='age', hue='income', kde=True, palette='viridis', ax=a
```



In [72]:
```
px.histogram(df, x='capital.gain', color="income", barmode='group', title='Inco
```

In [73]:
```
px.histogram(df, x='capital.loss', color="income", barmode='group', title='Inco
```

In [74]:
```
df['capital_diff'] = df['capital.gain'] - df['capital.loss']
df['capital_diff'] = pd.cut(df['capital_diff'], bins = [-5000, 5000, 100000], ]
df['capital_diff'] = df['capital_diff'].astype('object')
df.drop(['capital.gain'], axis = 1, inplace = True)
df.drop(['capital.loss'], axis = 1, inplace = True)
```

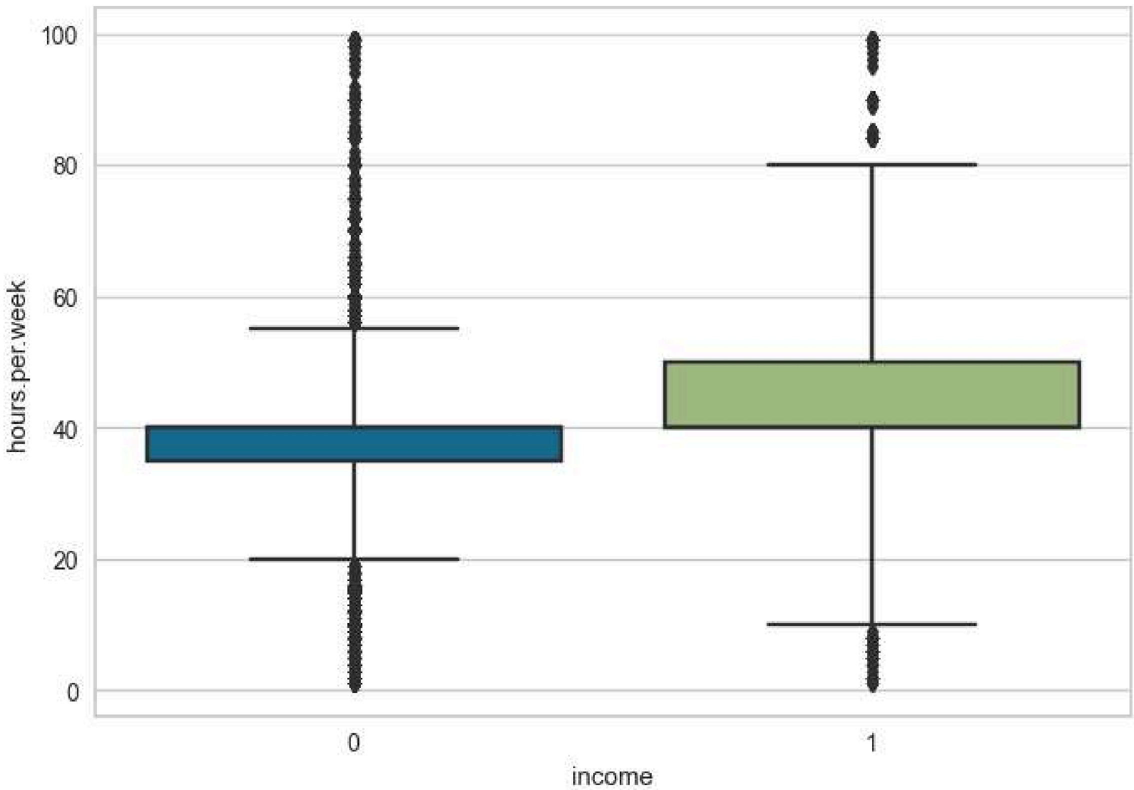
Purpose: To combine the capital.gain (capital gain) and capital.loss (capital loss) columns into a single column to calculate the net capital gain.

Result: A new column named capital_diff is created.

In [75]:
```
px.histogram(df, x='capital_diff', color="income", barmode='group', title='Inco
```

In [76]:
```
px.histogram(df, x='hours.per.week', color="income", barmode='group', title='In
```

In [43]:
```
sns.boxplot(data=df,y="hours.per.week",x='income', whis=3);
```



In [44]:
```
outliers = df[df['hours.per.week'] > 80]
outliers_income_counts = outliers['income'].value_counts()
outliers_income_counts
```

Out[44]:
```
income
0    145
1     63
Name: count, dtype: int64
```

In [45]:
```
outliers = df[df['hours.per.week'] < 15]
outliers_income_counts = outliers['income'].value_counts()
outliers_income_counts
```

Out[45]:
```
income
0    892
1     81
Name: count, dtype: int64
```

In [77]:
```
df = df[~((df["hours.per.week"] > 80) | (df["hours.per.week"] < 15))]
```

The code segments are used to analyze the income status of individuals with extremely high or low weekly working hours and to remove these outliers from the dataset.

In [78]:
```
df.drop(['fnlwgt'], axis = 1, inplace = True)
```

fnlwgt: As a result of the analysis, the effect of fnlwgt on the model is almost negligible. Therefore, it was excluded from the data.

In [48]:
```
df.shape
```

Out[48]:
```
(31356, 13)
```

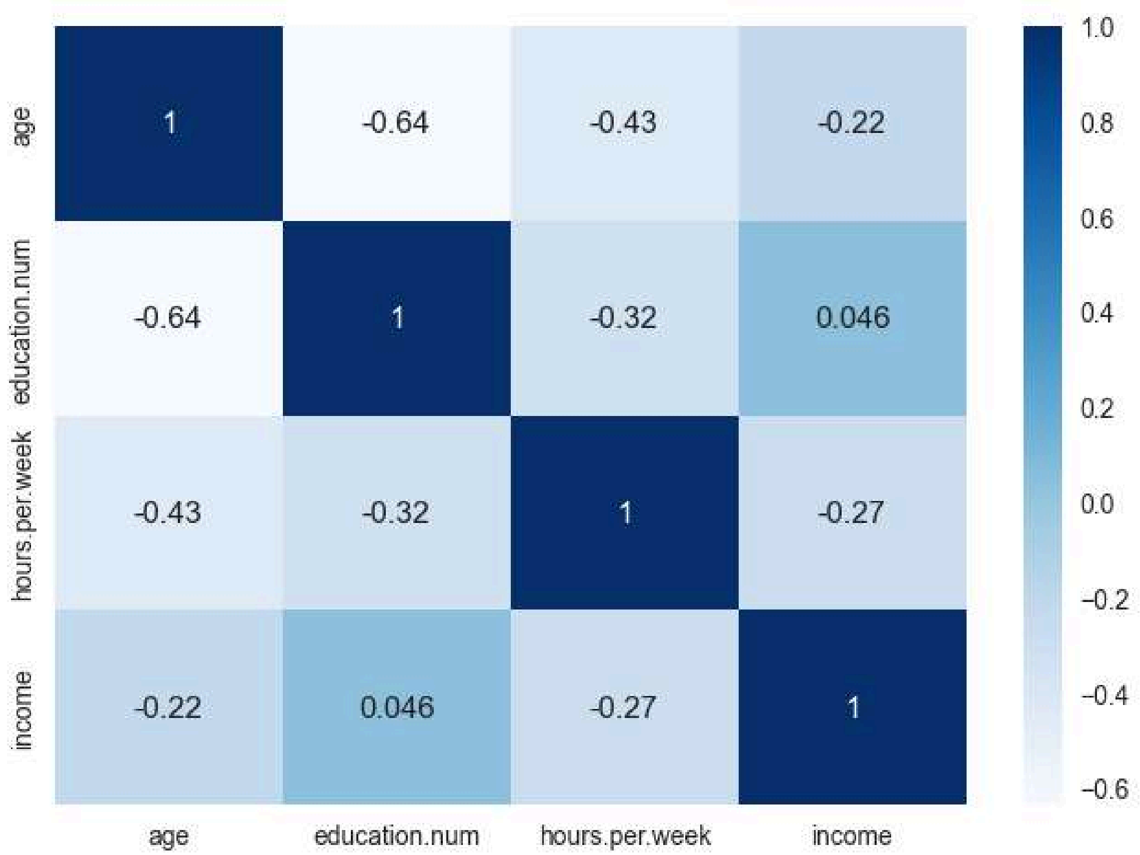
In [49]:
```
df.columns
```

Out[49]:
```
Index(['age', 'workclass', 'education', 'education.num', 'marital.status',
       'occupation', 'relationship', 'race', 'sex', 'hours.per.week',
       'native.country', 'income', 'capital_diff'],
      dtype='object')
```

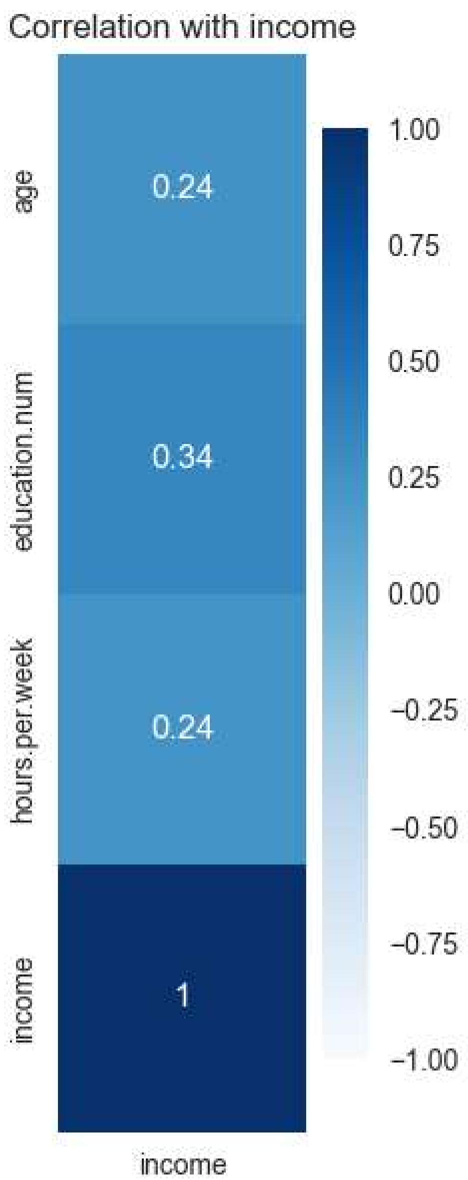
Correlation

In [79]:
```
numeric_df = df.select_dtypes(include=['number'])
corr_matrix = numeric_df.corr()
```

In [86]:
```
sns.heatmap(corr_matrix .corr(), annot=True, cmap="Blues");
```



In [87]:
```
def plot_target_correlation_heatmap(df, target_variable):
    df_numeric = df.select_dtypes(include=[np.number])
    df_corr_target = df_numeric.corr()

    plt.figure(figsize=(2, 7))
    sns.heatmap(df_corr_target[[target_variable]], annot=True, vmin=-1, vmax=1
    plt.title(f'Correlation with {target_variable}')
    plt.show()
plot_target_correlation_heatmap(df, 'income')
```



Multicollinearity

In [52]:
```
def color_correlation1(val):
    """
    Takes a scalar and returns a string with
    the css property in a variety of color scales
    for different correlations.
    """
    if val >= 0.6 and val < 0.99999 or val <= -0.6 and val > -0.99999:
        color = 'red'
    elif val < 0.6 and val >= 0.3 or val > -0.6 and val <= -0.3:
        color = 'blue'
    elif val == 1:
        color = 'green'
    else:
        color = 'black'
    return 'color: %s' % color

numeric_df = df.select_dtypes(include=[np.number])

numeric_df.corr().style.applymap(color_correlation1)
```

Out[52]:

	age	education.num	hours.per.week	income
age	1.000000	0.035414	0.110759	0.244210
education.num	0.035414	1.000000	0.163208	0.336660
hours.per.week	0.110759	0.163208	1.000000	0.241994
income	0.244210	0.336660	0.241994	1.000000

In [80]:
```
X = df.drop("income", axis=1)
y = df['income']
```

Models

Train | Test Split

In [81]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=42)
```

make_column_transformer

In [24]:
```
df.columns
```

Out[24]:
```
Index(['age', 'workclass', 'education', 'education.num', 'marital.status',
       'occupation', 'relationship', 'race', 'sex', 'hours.per.week',
       'native.country', 'income', 'capital_diff'],
      dtype='object')
```

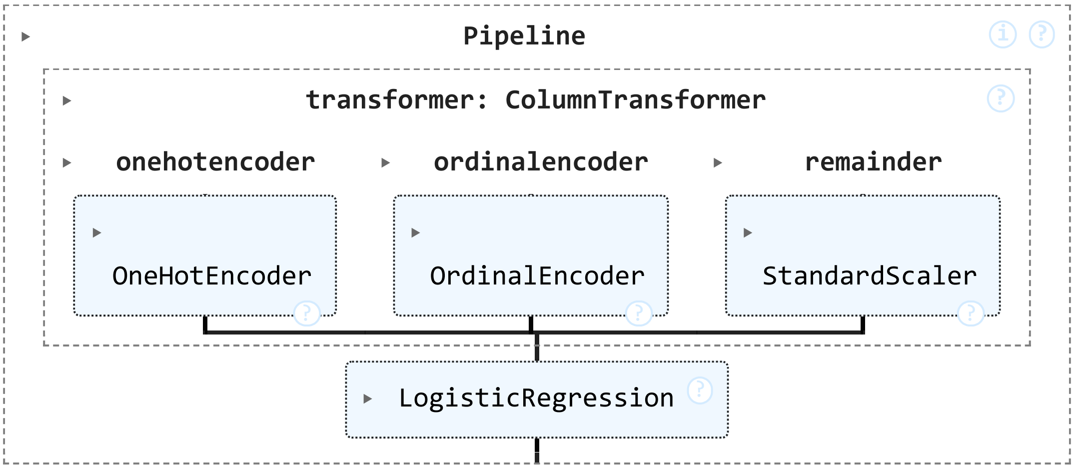
In [82]:
```
cat_onehot = [
    'workclass', 'occupation', 'relationship', 'race', 'sex', 'native.country',
    'marital.status'
]
cat_ordinal = ['education', 'capital_diff']

cat_for_edu = [
    'Preschool', 'elementary_school', 'secondary_school', 'HS-grad',
    'Some-college', 'Assoc', 'Bachelors', 'Masters', 'Prof-school', 'Doctorate
]
cat_for_capdiff = ['Low', 'High']
```

In [83]:
```
column_trans = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore", sparse_output=False), cat_onehot),
    (OrdinalEncoder(categories=[cat_for_edu, cat_for_capdiff]), cat_ordinal),
    remainder=StandardScaler())
```

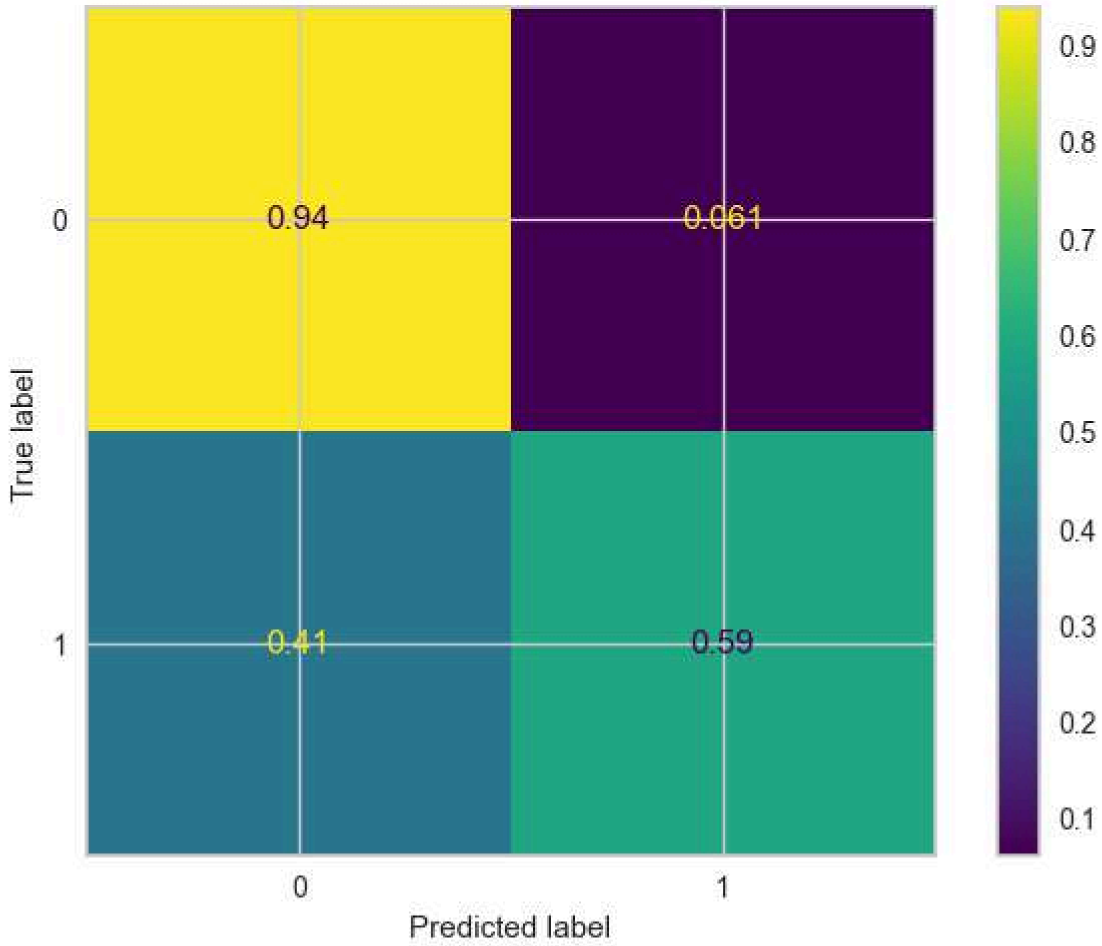
Logistic Regression Model

In [84]:
```
operations = [("transformer", column_trans), ("logistic", LogisticRegression(ma

pipe_model = Pipeline(steps=operations)


pipe_model.fit(X_train, y_train)
```

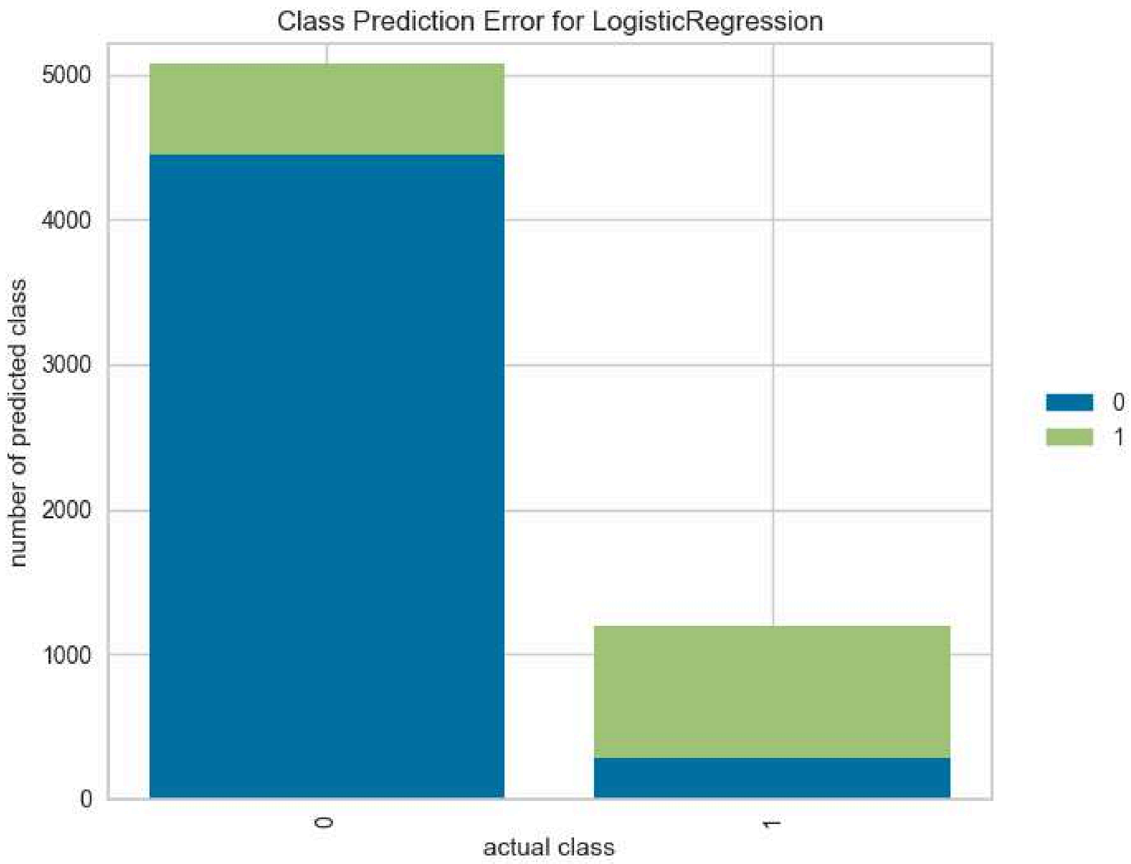
Out[84]:



In [85]:
```
ConfusionMatrixDisplay.from_estimator(pipe_model,
                                      X_test,
                                      y_test,
                                      normalize='true');
```



In [86]:
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(pipe_model)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



In [87]:
```
def eval_metric(model, X_train, y_train, X_test, y_test,i):
    y_train_pred = model.predict(X_train)
    y_pred = model.predict(X_test)
    print(f"{i} Test_Set")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print()
    print(f"{i} Train_Set")
    print(confusion_matrix(y_train, y_train_pred))
    print(classification_report(y_train, y_train_pred))
```

In [88]:
```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "logistic")
```

```
logistic Test_Set
[[4443  290]
 [ 636  903]]
              precision    recall  f1-score   support

           0       0.87      0.94      0.91      4733
           1       0.76      0.59      0.66      1539

    accuracy                           0.85      6272
   macro avg       0.82      0.76      0.78      6272
weighted avg       0.85      0.85      0.85      6272


logistic Train_Set
[[17632  1296]
 [ 2528  3628]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90     18928
           1       0.74      0.59      0.65      6156

    accuracy                           0.85     25084
   macro avg       0.81      0.76      0.78     25084
weighted avg       0.84      0.85      0.84     25084
```

Cross Validate

In [113…

```
operations = [("transformer", column_trans), ("logistic", LogisticRegression(ma

pipecv_model = Pipeline(steps=operations)

cv = StratifiedKFold(n_splits=10)

scores = cross_validate(pipecv_model,
                        X_train,
                        y_train,
                        scoring=["accuracy", "precision", "recall", "f1"],
                        cv=cv,
                        return_train_score = True)
df_scores = pd.DataFrame(scores, index=range(1,11))
df_scores.mean()[2:]
```

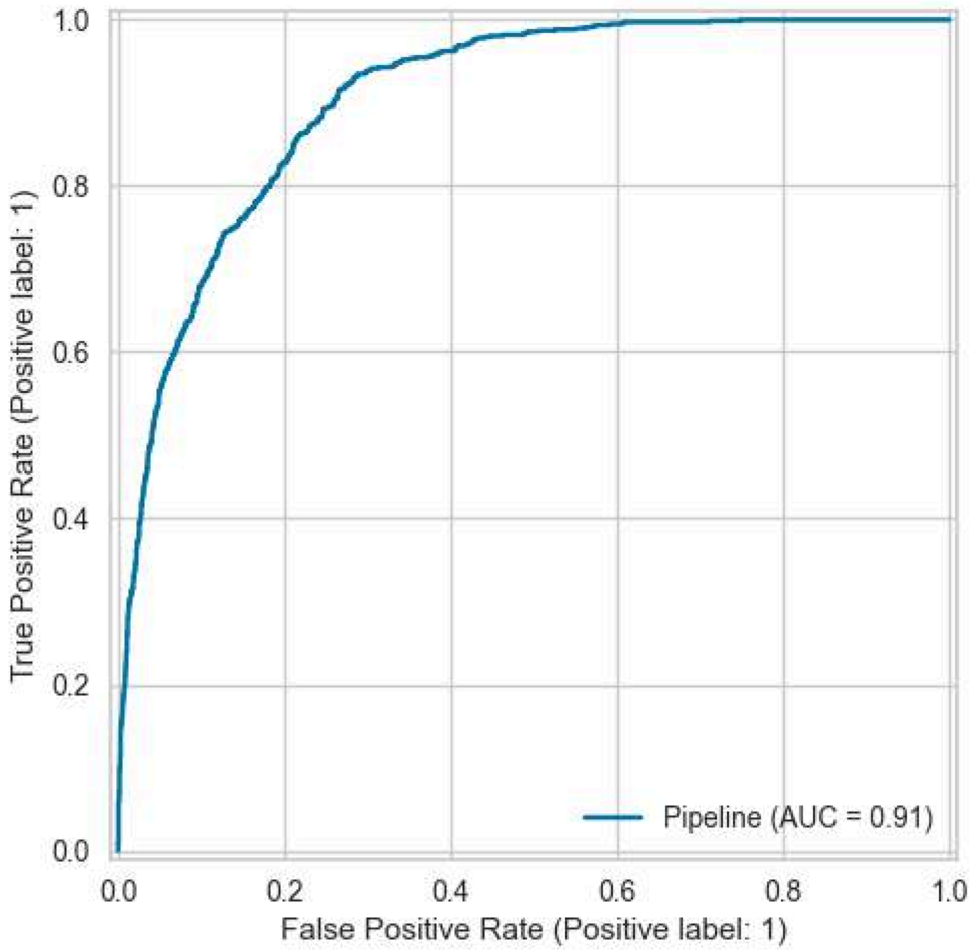
Out[113…

```
test_accuracy      0.846596
train_accuracy     0.847543
test_precision     0.734494
train_precision    0.736169
test_recall        0.588371
train_recall       0.590355
test_f1            0.653002
train_f1           0.655246
dtype: float64
```

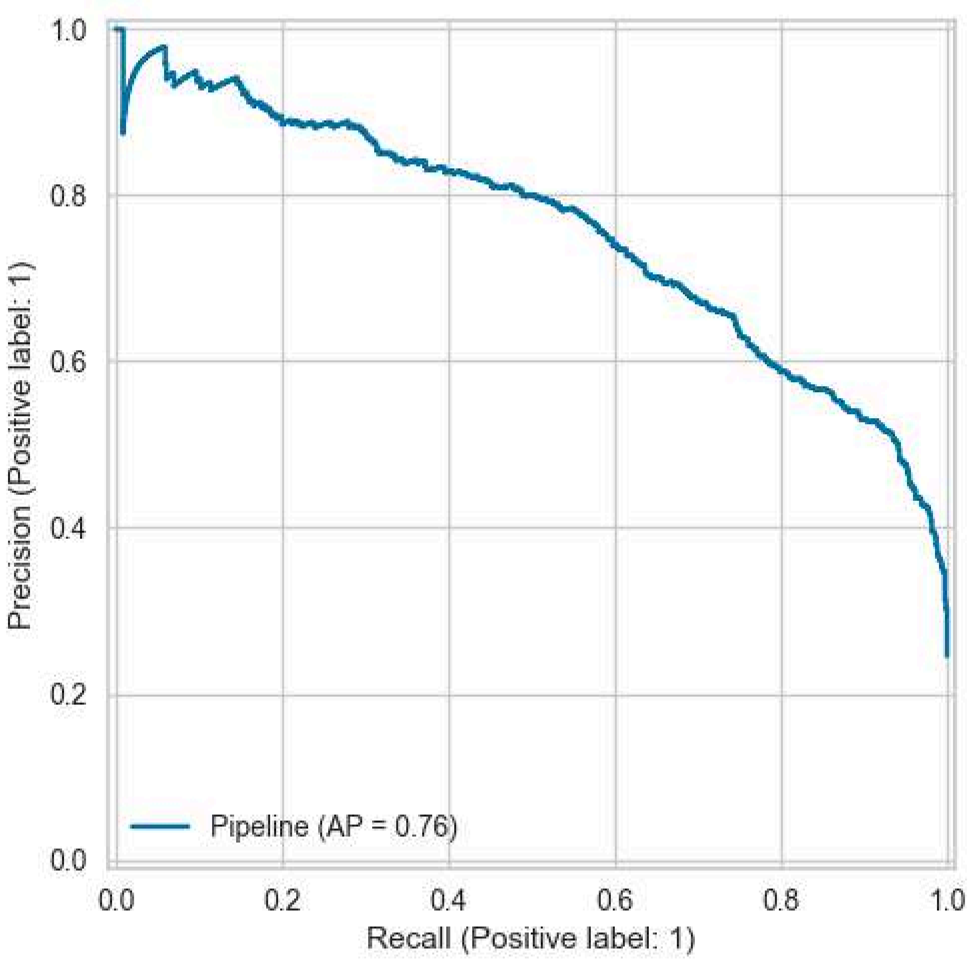
Precision Recall Curve and Roc Curve Display

In [30]:

```
RocCurveDisplay.from_estimator(pipe_model, X_test, y_test);
```



In [31]:
```
PrecisionRecallDisplay.from_estimator(pipe_model, X_test, y_test);
```



GridSearchCV

param_grid = [{ "logistic__penalty" : ['l1', 'l2'], "logistic__C" : [0.01, 0.05,0.03, 0.1, 1], "logistic__class_weight": ["balanced", None] , "logistic__solver": ['liblinear', 'saga', 'lbfgs'], "logistic__max_iter": [1000, 2000] }]

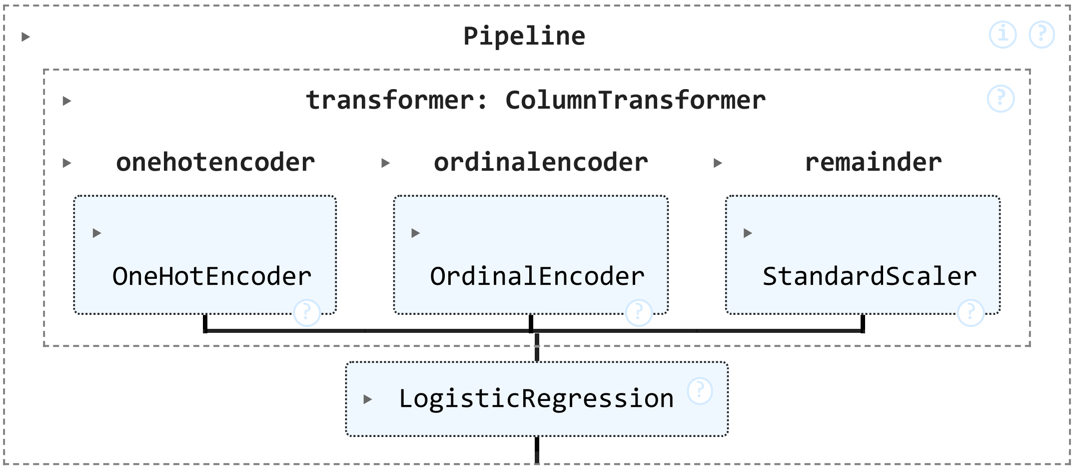
Many grids of money have been tried. Finally, the following features were identified.

In [89]:
```
operations = [("transformer", column_trans), ("logistic", LogisticRegression(ra

log_model = Pipeline(steps=operations)

param_grid = [
    {
        "logistic__penalty" : ['l1'],
        "logistic__C" : [0.03],
        "logistic__class_weight": ["balanced"] ,
        "logistic__solver": ['saga'],
        "logistic__max_iter": [1000]
    }
]
cv = StratifiedKFold(n_splits = 10)

grid_model = GridSearchCV(estimator=log_model,
                          param_grid=param_grid,
                          cv=cv,
                          scoring = "f1",
```

```
                        n_jobs = -1,
                        return_train_score=True).fit(X_train, y_train)
```

In [90]:
```
grid_model.best_estimator_
```

Out[90]:



In [91]:
```
grid_model.best_score_
```

Out[91]:
```
0.683264633932356
```

In [92]:
```
grid_model.best_index_
```

Out[92]:
```
0
```

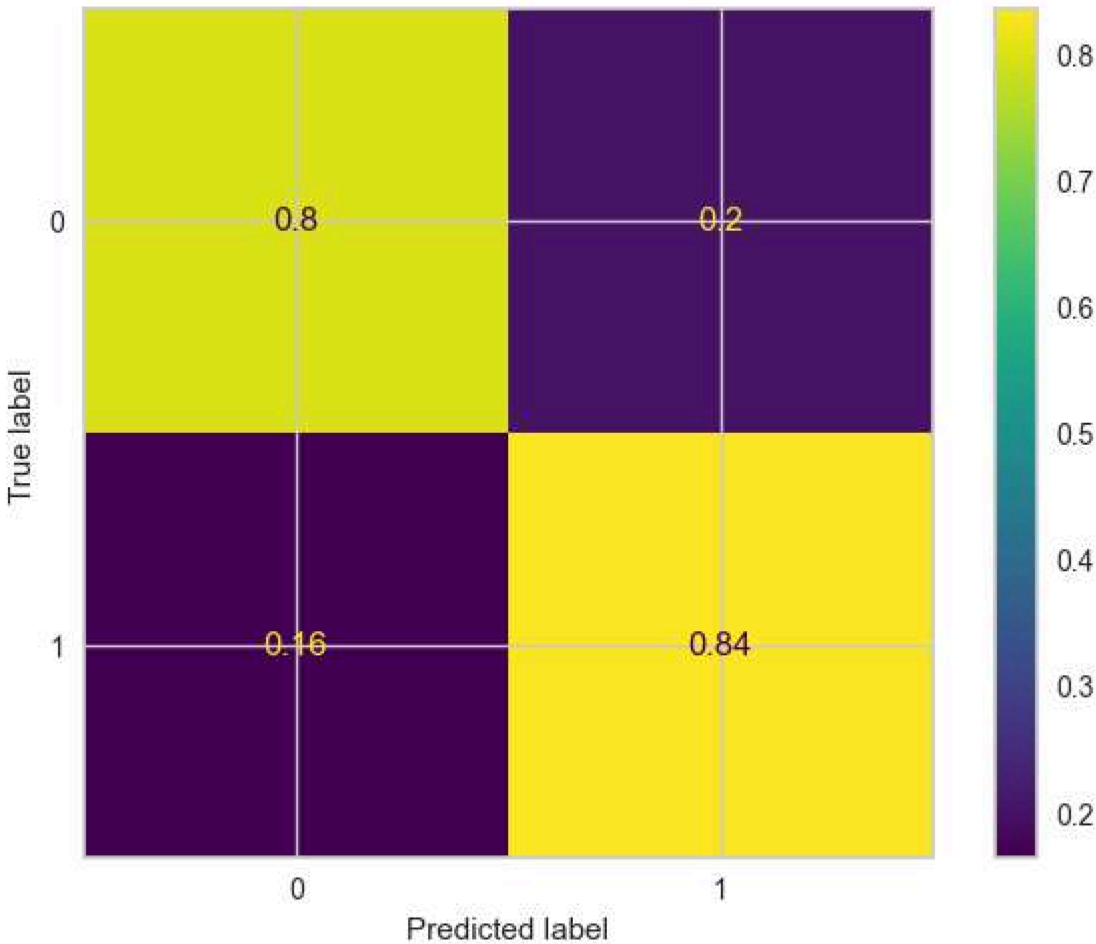
In [93]:
```
pd.DataFrame(grid_model.cv_results_).loc[0, ["mean_test_score", "mean_train_sco
```

Out[93]:
```
mean_test_score     0.683265
mean_train_score    0.682596
Name: 0, dtype: object
```

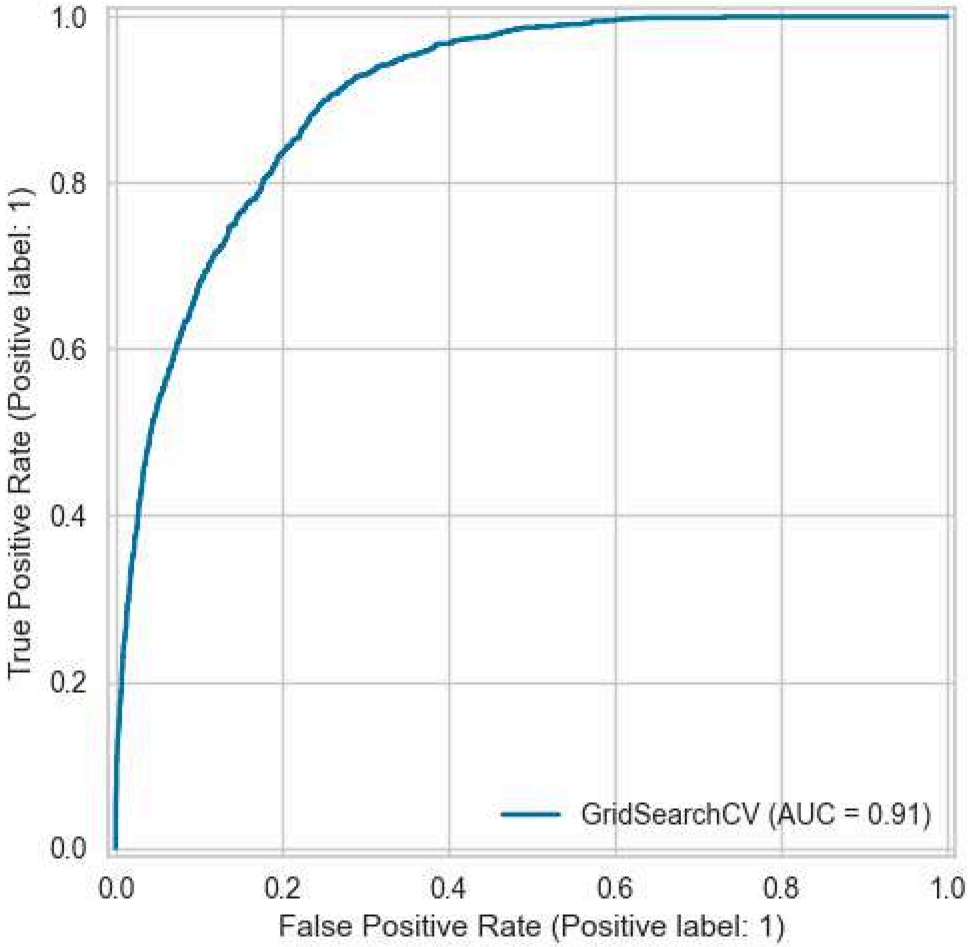
In [94]:
```
y_pred = grid_model.predict(X_test)
y_pred_proba = grid_model.predict_proba(X_test)

log_f1 = f1_score(y_test, y_pred)

log_recall = recall_score(y_test, y_pred)

log_auc = roc_auc_score(y_test, y_pred)

precision, recall, _ = precision_recall_curve(y_test, grid_model.predict_proba(
log_prc = auc(recall, precision)

log_grid_model = eval_metric(grid_model, X_train, y_train, X_test, y_test,"logi
log_grid_model
```

```
logisticgrid Test_Set
[[3783  950]
 [ 253 1286]]
              precision    recall  f1-score   support

           0       0.94      0.80      0.86      4733
           1       0.58      0.84      0.68      1539

    accuracy                           0.81      6272
   macro avg       0.76      0.82      0.77      6272
weighted avg       0.85      0.81      0.82      6272


logisticgrid Train_Set
[[15047  3881]
 [  952  5204]]
              precision    recall  f1-score   support

           0       0.94      0.79      0.86     18928
           1       0.57      0.85      0.68      6156

    accuracy                           0.81     25084
   macro avg       0.76      0.82      0.77     25084
weighted avg       0.85      0.81      0.82     25084
```

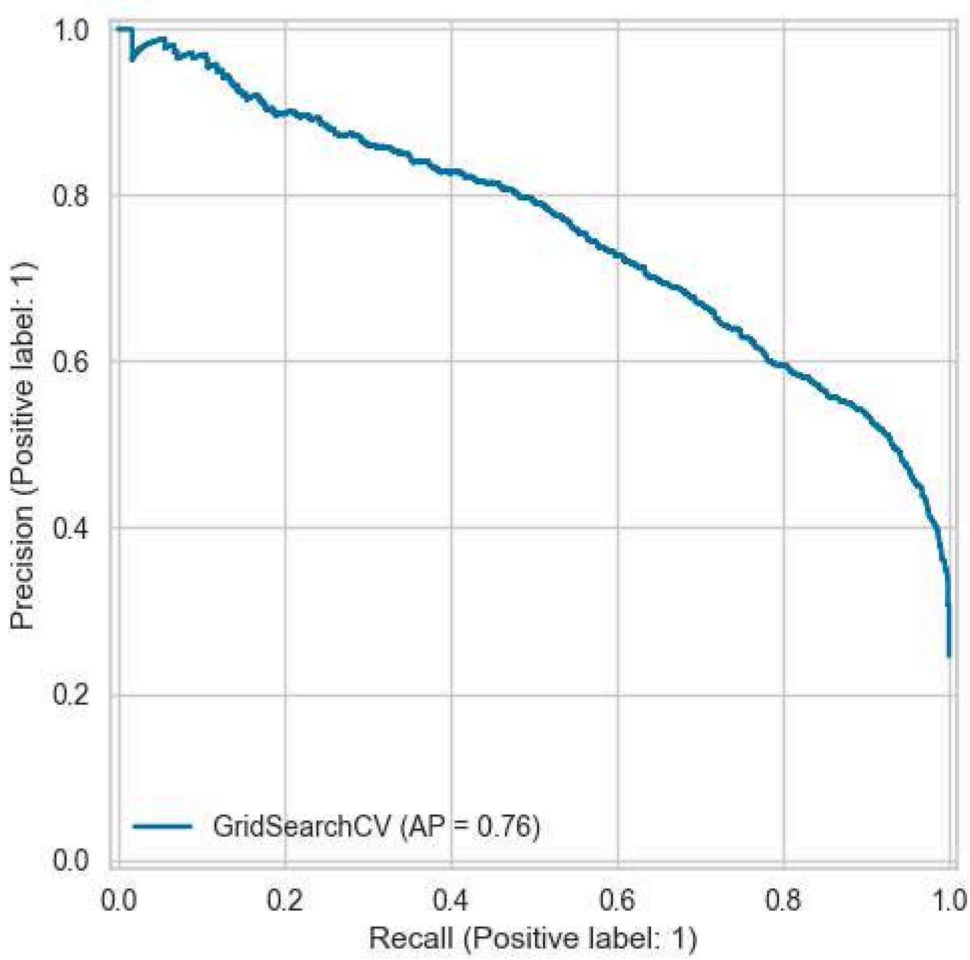
In [95]:
```
log_grid_matrix = ConfusionMatrixDisplay.from_estimator(grid_model,
                                                        X_test,
                                                        y_test,
                                                        normalize='true');
```



In [96]:
```
RocCurveDisplay.from_estimator(grid_model, X_test, y_test);
```



In [97]:
```
PrecisionRecallDisplay.from_estimator(grid_model, X_test, y_test);
```



KNN Model

In [39]:
```
operations = [("transformer", column_trans), ("knn", KNeighborsClassifier())]

pipe_model = Pipeline(steps=operations)

pipe_model.fit(X_train, y_train)
```

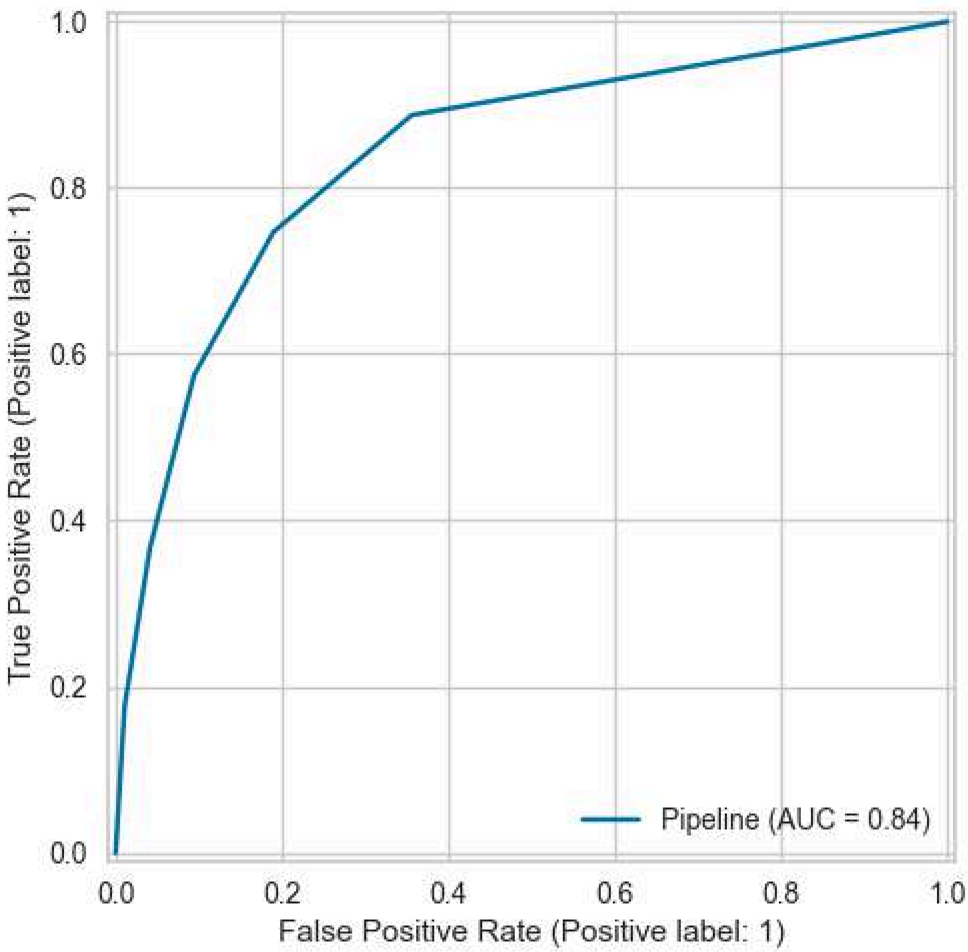
Out[39]:
```
Pipeline
  transformer: ColumnTransformer
    onehotencoder      ordinalencoder      remainder
    OneHotEncoder      OrdinalEncoder      StandardScaler
  KNeighborsClassifier
```

In [164…
```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "knn")
```

```
knn Test_Set
[[4286  447]
 [ 655  884]]
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      4733
           1       0.66      0.57      0.62      1539

    accuracy                           0.82      6272
   macro avg       0.77      0.74      0.75      6272
weighted avg       0.82      0.82      0.82      6272


knn Train_Set
[[17746  1182]
 [ 1881  4275]]
              precision    recall  f1-score   support

           0       0.90      0.94      0.92     18928
           1       0.78      0.69      0.74      6156

    accuracy                           0.88     25084
   macro avg       0.84      0.82      0.83     25084
weighted avg       0.87      0.88      0.88     25084
```

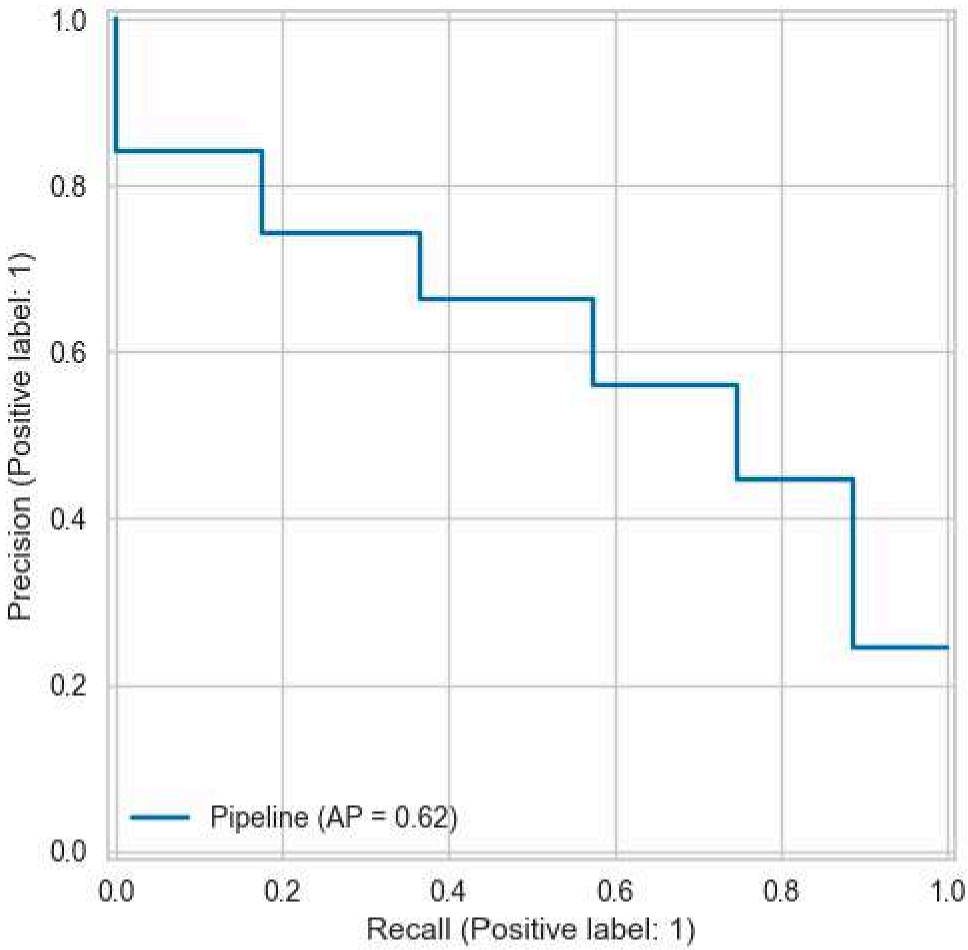
In [168…

```
RocCurveDisplay.from_estimator(pipe_model, X_test, y_test);
```



In [169…

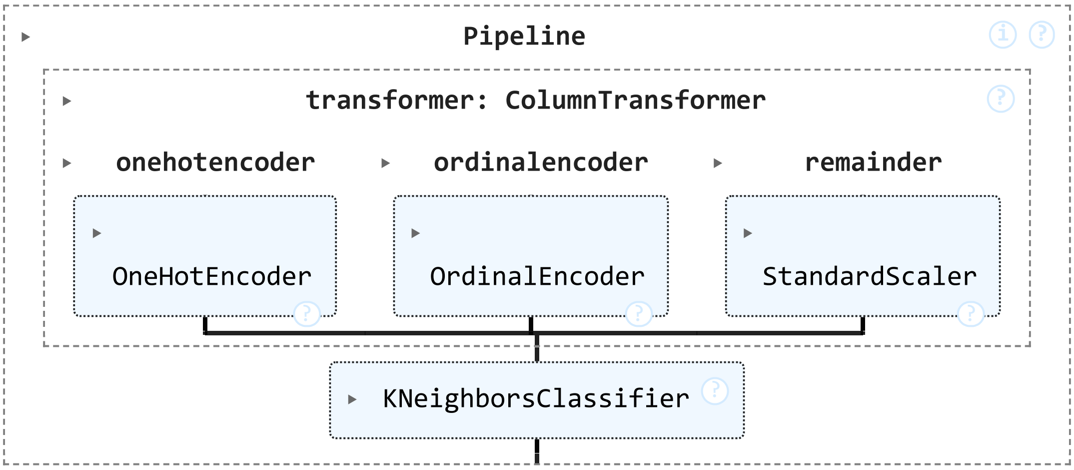
```
PrecisionRecallDisplay.from_estimator(pipe_model, X_test, y_test);
```



Elbow Method for Choosing Reasonable K Values

In [98]:
```
operations = [("transformer", column_trans), ("knn", KNeighborsClassifier())]

pipe_model = Pipeline(steps=operations)

pipe_model.fit(X_train, y_train)
```

Out[98]:



In [172…
```
test_error_rates = []

for k in range(1, 10):

    operations = [("transformer", column_trans), ("knn", KNeighborsClassifier(n
```

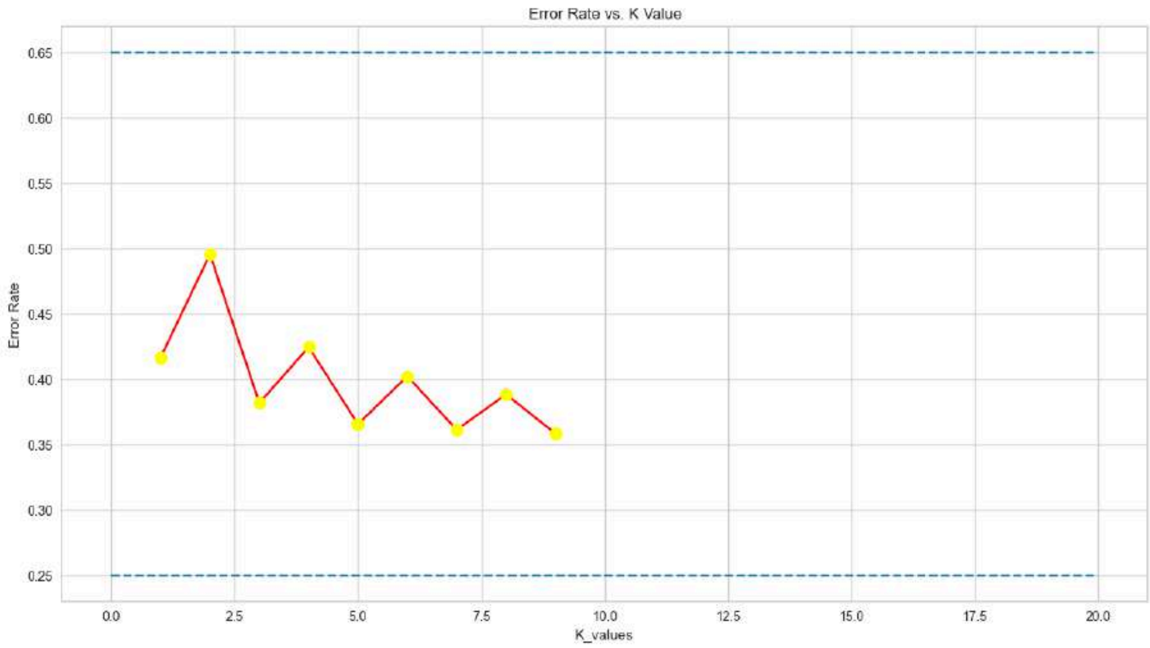
```
    knn_pipe_model = Pipeline(steps=operations)

    scores = cross_validate(knn_pipe_model, X_train, y_train, scoring = ['f1'].

    f1_mean = scores["test_f1"].mean()

    test_error = 1 - f1_mean

    test_error_rates.append(test_error)
```

In [174…

```
plt.figure(figsize=(15, 8))
plt.plot(range(1, 10),
         test_error_rates,
         color='red',
         marker='o',
         markerfacecolor='yellow',
         markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.25, xmin=0, xmax=20, colors='b', linestyles="--")
plt.hlines(y=0.65, xmin=0, xmax=20, colors='b', linestyles="--")
```

Out[174…

```
<matplotlib.collections.LineCollection at 0x2604749d090>
```



Overfiting and underfiting control for k values

In [175…

```
test_error_rates = []
train_error_rates = []

for k in range(1, 10):

    operations = [("transformer", column_trans), ("knn", KNeighborsClassifier(r

    knn_pipe_model = Pipeline(steps=operations)

    knn_pipe_model.fit(X_train, y_train)
```

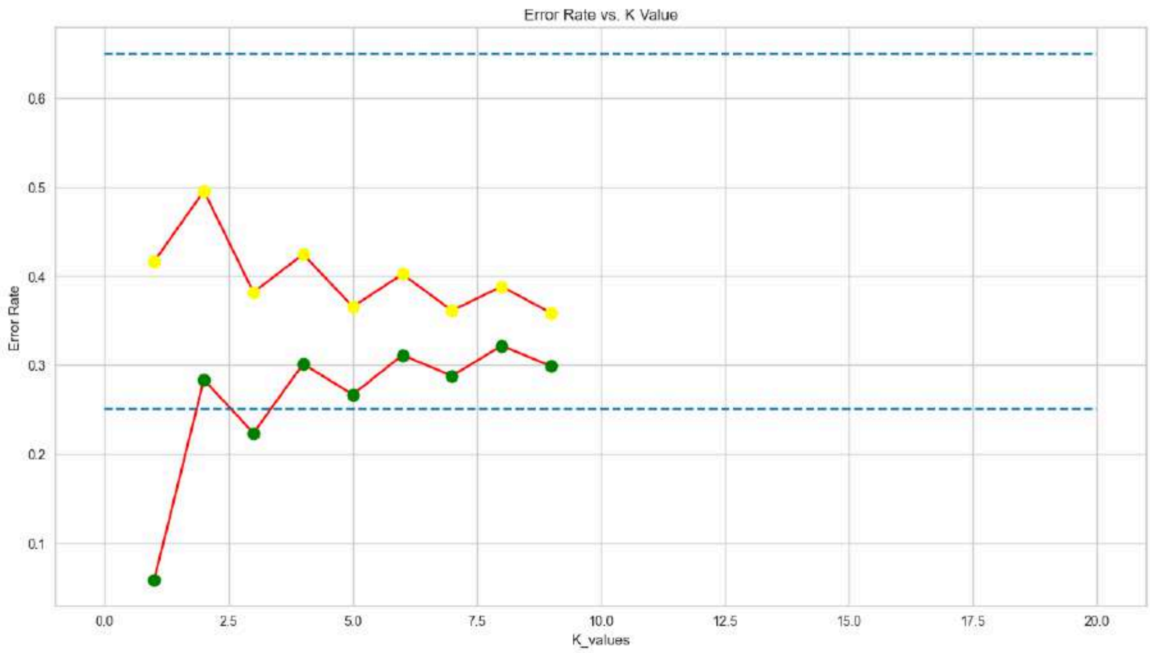
```
    scores = cross_validate(knn_pipe_model, X_train, y_train, scoring = ['f1'],

    f1_test_mean = scores["test_f1"].mean()
    f1_train_mean = scores["train_f1"].mean()


    test_error = 1 - f1_test_mean
    train_error = 1 -f1_train_mean
    test_error_rates.append(test_error)
    train_error_rates.append(train_error)
```

In [176…

```
plt.figure(figsize=(15, 8))
plt.plot(range(1, 10),
         test_error_rates,
         color='red',
         marker='o',
         markerfacecolor='yellow',
         markersize=10)

plt.plot(range(1, 10),
         train_error_rates,
         color='red',
         marker='o',
         markerfacecolor='green',
         markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.25, xmin=0, xmax=20, colors='b', linestyles="--")
plt.hlines(y=0.65, xmin=0, xmax=20, colors='b', linestyles="--")
```

Out[176…

```
<matplotlib.collections.LineCollection at 0x260475d7150>
```



In [177…

```
k_list = [3, 5, 7]

for i in k_list:
    operations = [("transformer", column_trans), ("knn", KNeighborsClassifier(n
    knn = Pipeline(steps=operations)
```

```
    knn.fit(X_train, y_train)
    print(f'WITH K={i}\n')
    eval_metric(knn, X_train, y_train, X_test, y_test, "knn_elbow")
```

```
WITH K=3

knn_elbow Test_Set
[[4236  497]
 [ 684  855]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.88      4733
           1       0.63      0.56      0.59      1539

    accuracy                           0.81      6272
   macro avg       0.75      0.73      0.73      6272
weighted avg       0.80      0.81      0.81      6272


knn_elbow Train_Set
[[17848  1080]
 [ 1568  4588]]
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     18928
           1       0.81      0.75      0.78      6156

    accuracy                           0.89     25084
   macro avg       0.86      0.84      0.85     25084
weighted avg       0.89      0.89      0.89     25084

WITH K=5

knn_elbow Test_Set
[[4286  447]
 [ 655  884]]
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      4733
           1       0.66      0.57      0.62      1539

    accuracy                           0.82      6272
   macro avg       0.77      0.74      0.75      6272
weighted avg       0.82      0.82      0.82      6272


knn_elbow Train_Set
[[17746  1182]
 [ 1881  4275]]
              precision    recall  f1-score   support

           0       0.90      0.94      0.92     18928
           1       0.78      0.69      0.74      6156

    accuracy                           0.88     25084
   macro avg       0.84      0.82      0.83     25084
weighted avg       0.87      0.88      0.88     25084

WITH K=7

knn_elbow Test_Set
[[4315  418]
 [ 647  892]]
              precision    recall  f1-score   support
```

```
           0       0.87      0.91      0.89      4733
           1       0.68      0.58      0.63      1539

    accuracy                           0.83      6272
   macro avg       0.78      0.75      0.76      6272
weighted avg       0.82      0.83      0.83      6272


knn_elbow Train_Set
[[17663  1265]
 [ 2017  4139]]
              precision    recall  f1-score   support

           0       0.90      0.93      0.91     18928
           1       0.77      0.67      0.72      6156

    accuracy                           0.87     25084
   macro avg       0.83      0.80      0.82     25084
weighted avg       0.87      0.87      0.87     25084
```

Cross Validate For Optimal K Value

In [178…

```
operations =   operations = [("transformer", column_trans), ("knn", KNeighbors(

model = Pipeline(steps=operations)

scores = cross_validate(model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        cv=10,
                        return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores.mean()[2:]
```

Out[178…

```
test_accuracy      0.834077
train_accuracy     0.868296
test_precision     0.684224
train_precision    0.763656
test_recall        0.602503
train_recall       0.671017
test_f1            0.640533
train_f1           0.714341
dtype: float64
```

Gridsearch Method for Choosing Reasonable K Values

In [100…

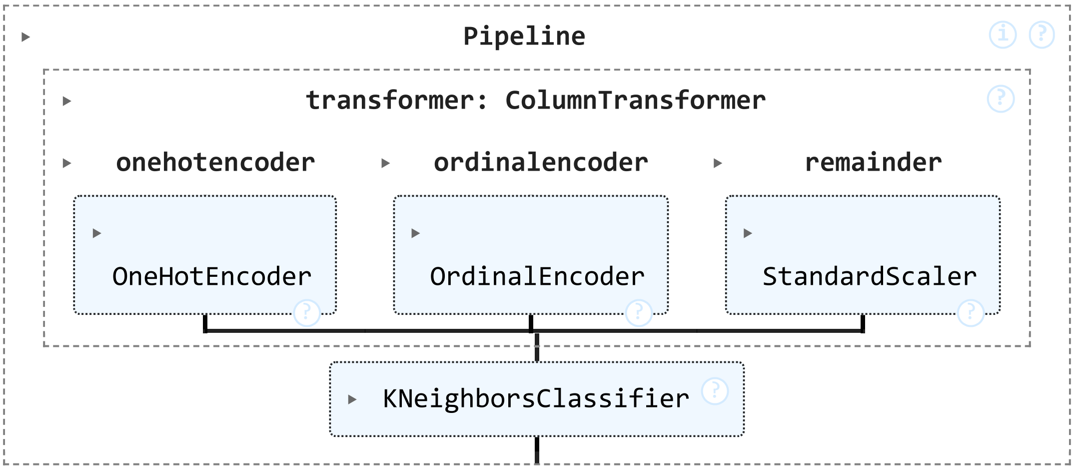
```
operations = [("transformer", column_trans), ("knn", KNeighborsClassifier())]
knn_model = Pipeline(steps=operations)
```

Many grids of money have been tried. Finally, the following features were identified.Tried values up to k_values = 30.

In [101…
```
param_grid = [
    {
        "knn__n_neighbors": [19],
        "knn__metric": ['euclidean'],
        "knn__weights": ['uniform']
    }
]

knn_grid_model = GridSearchCV(knn_model,
                              param_grid,
                              scoring='f1',
                              cv=5,
                              return_train_score=True,
                              n_jobs=-1).fit(X_train, y_train)
```

In [102…
```
knn_grid_model.best_estimator_
```

Out[102…



In [103…
```
knn_grid_model.best_index_
```

Out[103…
```
0
```

In [104…
```
pd.DataFrame(
    knn_grid_model.cv_results_).loc[0,["mean_test_score", "mean_train_score"]]
```

Out[104…
```
mean_test_score       0.6397
mean_train_score    0.675013
Name: 0, dtype: object
```

In [105…
```
knn_grid_model.best_score_
```

Out[105…
```
0.6396999259520801
```

In [106…
```
y_pred = knn_grid_model.predict(X_test)
y_pred_proba = knn_grid_model.predict_proba(X_test)

knn_f1 = f1_score(y_test, y_pred)

knn_recall = recall_score(y_test, y_pred)

knn_auc = roc_auc_score(y_test, y_pred)

precision, recall, _ = precision_recall_curve(y_test, knn_grid_model.predict_p
knn_prc = auc(recall, precision)
```

```
eval_metric(knn_grid_model, X_train, y_train, X_test, y_test, "knn_grid") #k=1!
```

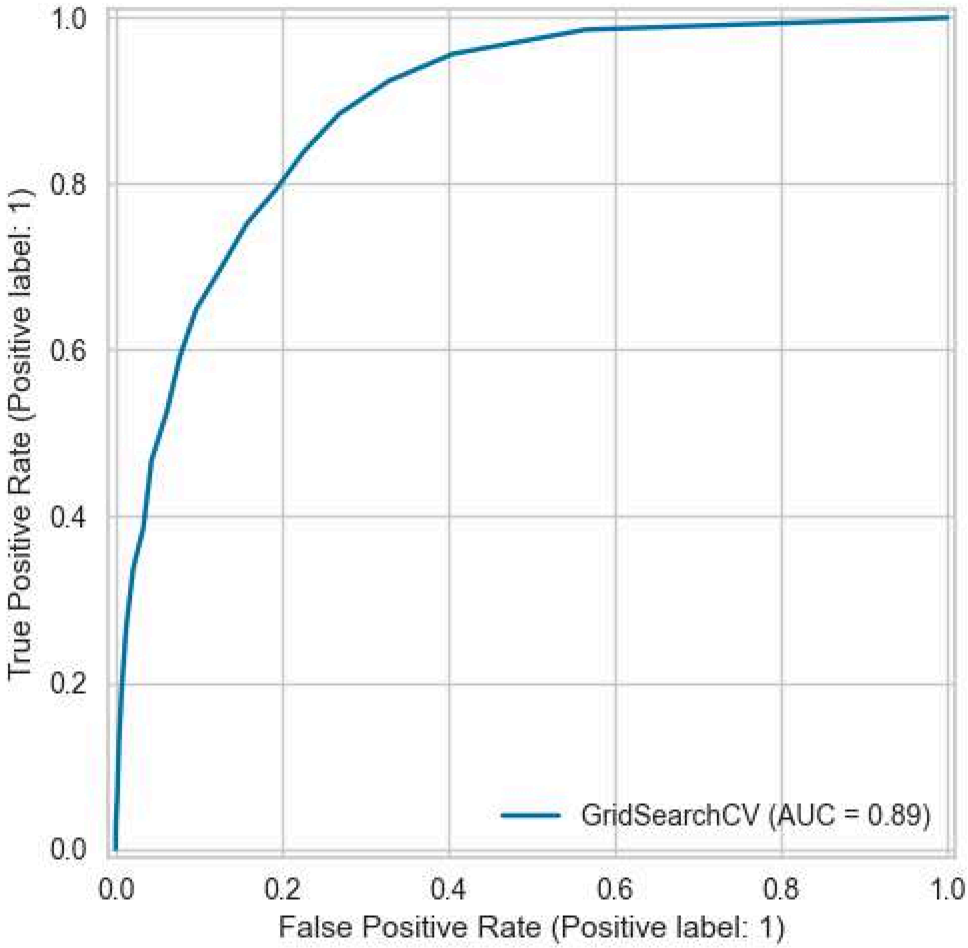
```
knn_grid Test_Set
[[4367  366]
 [ 628  911]]
              precision    recall  f1-score   support

           0       0.87      0.92      0.90      4733
           1       0.71      0.59      0.65      1539

    accuracy                           0.84      6272
   macro avg       0.79      0.76      0.77      6272
weighted avg       0.83      0.84      0.84      6272


knn_grid Train_Set
[[17492  1436]
 [ 2269  3887]]
              precision    recall  f1-score   support

           0       0.89      0.92      0.90     18928
           1       0.73      0.63      0.68      6156

    accuracy                           0.85     25084
   macro avg       0.81      0.78      0.79     25084
weighted avg       0.85      0.85      0.85     25084
```

As a result of the values we gave to K, the tests did not improve, but we prevented overfitting and found more reliable results

Precision Recall Curve and Roc Curve Display

In [190…

```
RocCurveDisplay.from_estimator(knn_grid_model, X_test, y_test);
```



In [191…

```
y_pred_proba = knn.predict_proba(X_test)
roc_auc_score(y_test, y_pred_proba[:,1])
```

Out[191…

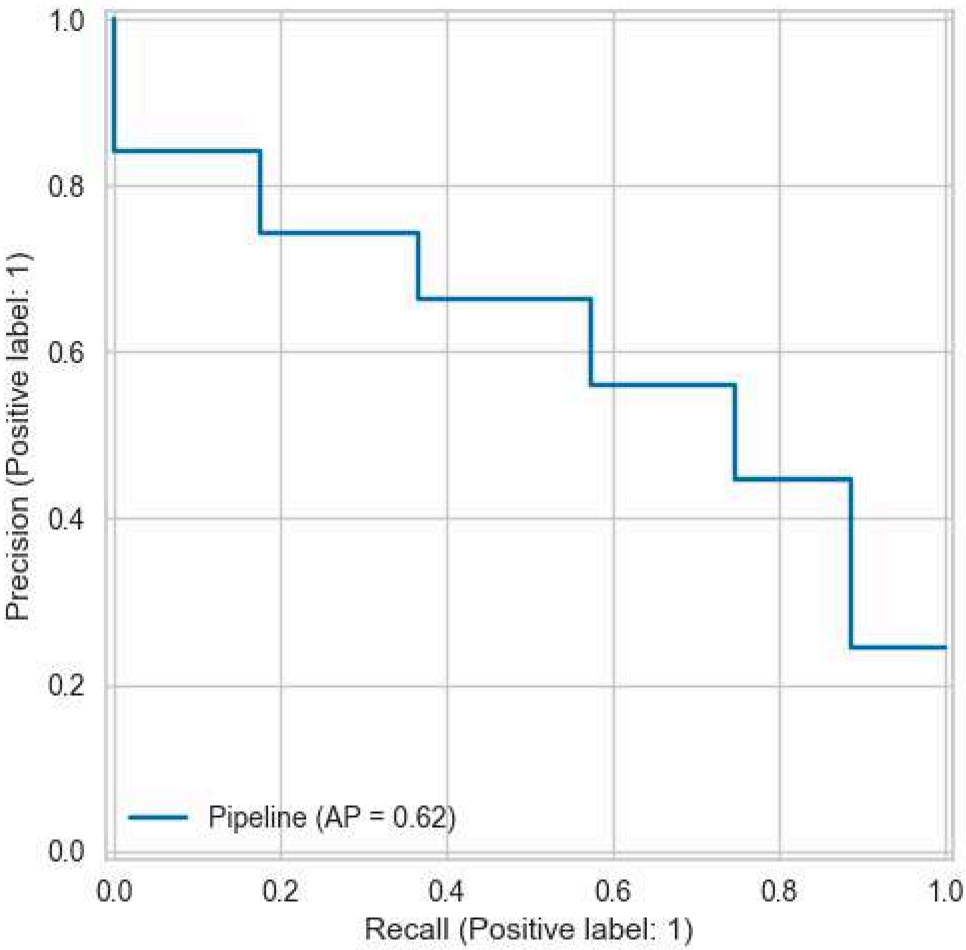
```
0.859903581601922
```

In [192…

```
PrecisionRecallDisplay.from_estimator(pipe_model, X_test, y_test)
```

Out[192…

```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x2604
727de50>
```

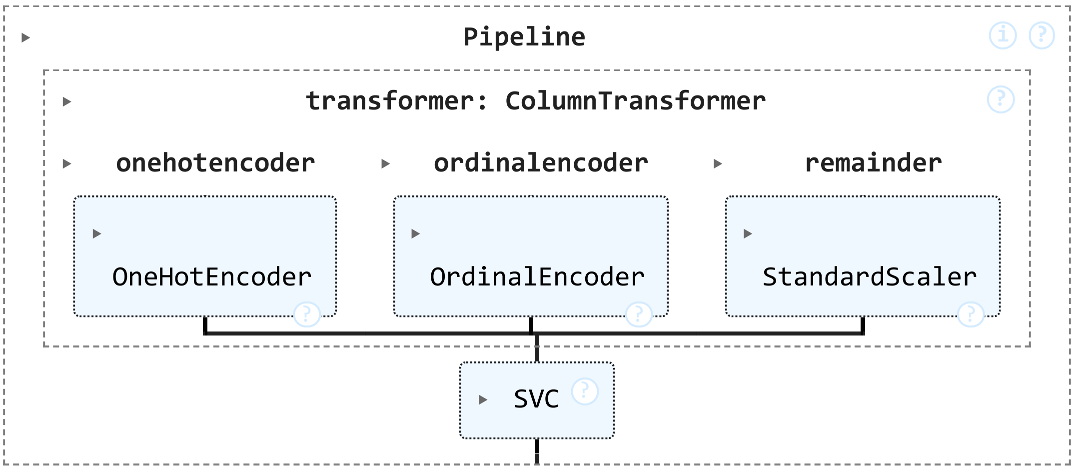


SVM Model

In [107…

```
operations = [("transformer", column_trans),("SVC", SVC(random_state=42))]
pipe_model = Pipeline(steps=operations)
pipe_model.fit(X_train, y_train)
```

Out[107…



Model Performance

In [194…

```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "svm")
```

```
svm Test_Set
[[4498  235]
 [ 692  847]]
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      4733
           1       0.78      0.55      0.65      1539

    accuracy                           0.85      6272
   macro avg       0.82      0.75      0.78      6272
weighted avg       0.85      0.85      0.84      6272


svm Train_Set
[[17901  1027]
 [ 2756  3400]]
              precision    recall  f1-score   support

           0       0.87      0.95      0.90     18928
           1       0.77      0.55      0.64      6156

    accuracy                           0.85     25084
   macro avg       0.82      0.75      0.77     25084
weighted avg       0.84      0.85      0.84     25084
```

In [198…

```python
operations = [("transformer", column_trans), ("SVC", SVC(random_state=42))]

pipe_model = Pipeline(steps=operations)

cv = StratifiedKFold(n_splits=5)

scores = cross_validate(pipe_model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        cv=cv,
                        return_train_score=True,
                        n_jobs=-1)

df_scores = pd.DataFrame(scores, index=range(1, 6))
df_scores.mean()[2:]
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   5 | elapsed:  2.5min remaining:  1.7mi
n
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  6.7min finished
```

Out[198…

```
test_accuracy      0.847353
train_accuracy     0.849097
test_precision     0.764130
train_precision    0.769085
test_recall        0.546947
train_recall       0.550357
test_f1            0.637474
train_f1           0.641591
dtype: float64
```

GridsearchCV

param_grid = {'SVC__C': [0.01, 0.1, 1, 10, 100], 'SVC__gamma': ["scale", "auto", 0.001, 0.01, 0.1, 0.5], 'SVC__kernel': ['rbf', 'linear'],}

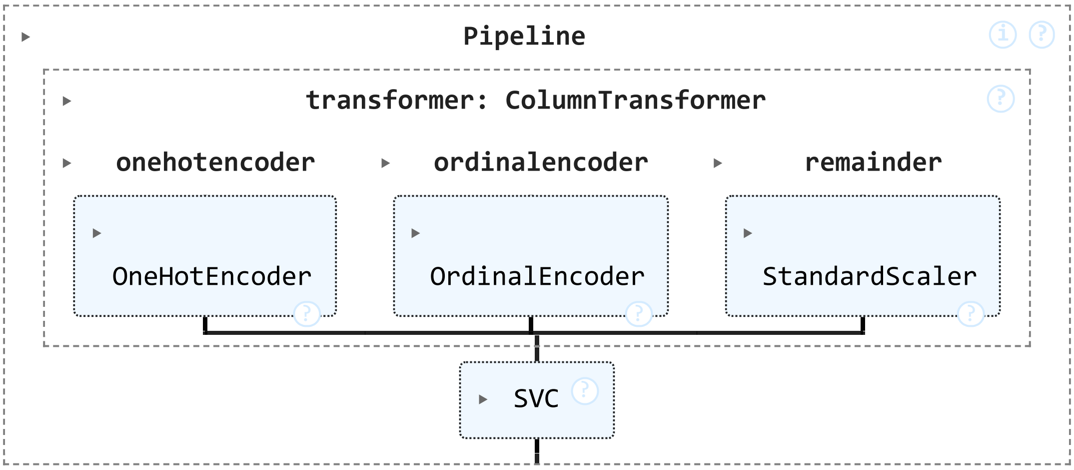
Many grids of money have been tried. Finally, the following features were identified.

In [108…
```
param_grid = {"SVC__C":[1],
              "SVC__gamma":[0.3],
              "SVC__kernel":["rbf"]}

operations = [("transformer", column_trans), ("SVC", SVC(class_weight="balance

svm_model_grid = GridSearchCV(pipe_model,
                              param_grid,
                              scoring="recall_macro",
                              cv=5,
                              return_train_score=True,
                              n_jobs=2,
                              verbose=2).fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

In [109…
```
svm_model_grid.best_estimator_
```

Out[109…



In [110…
```
svm_model_grid.best_index_
```

Out[110…
```
0
```

In [111…
```
pd.DataFrame(
    svm_model_grid.cv_results_).loc[0,
                                    ["mean_test_score", "mean_train_score"]]
```

Out[111…
```
mean_test_score     0.773176
mean_train_score    0.807849
Name: 0, dtype: object
```

In [112…
```
svm_model_grid.best_score_
```

Out[112…
```
0.7731760615298443
```

In [113…
```
y_pred = svm_model_grid.predict(X_test)
y_pred_proba = svm_model_grid.decision_function(X_test)

svm_f1 = f1_score(y_test, y_pred)
```

```
svm_recall = recall_score(y_test, y_pred)

svm_auc = roc_auc_score(y_test, y_pred)

precision, recall, _ = precision_recall_curve(y_test, svm_model_grid.decision_
svm_prc = auc(recall, precision)

eval_metric(svm_model_grid, X_train, y_train, X_test, y_test, "svm_grid")
```

```
svm_grid Test_Set
[[4409  324]
 [ 607  932]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      4733
           1       0.74      0.61      0.67      1539

    accuracy                           0.85      6272
   macro avg       0.81      0.77      0.79      6272
weighted avg       0.85      0.85      0.85      6272


svm_grid Train_Set
[[17835  1093]
 [ 2040  4116]]
              precision    recall  f1-score   support

           0       0.90      0.94      0.92     18928
           1       0.79      0.67      0.72      6156

    accuracy                           0.88     25084
   macro avg       0.84      0.81      0.82     25084
weighted avg       0.87      0.88      0.87     25084
```

In [114…

```
decision_function = svm_model_grid.decision_function(X_test)
average_precision_score(y_test, decision_function)
```

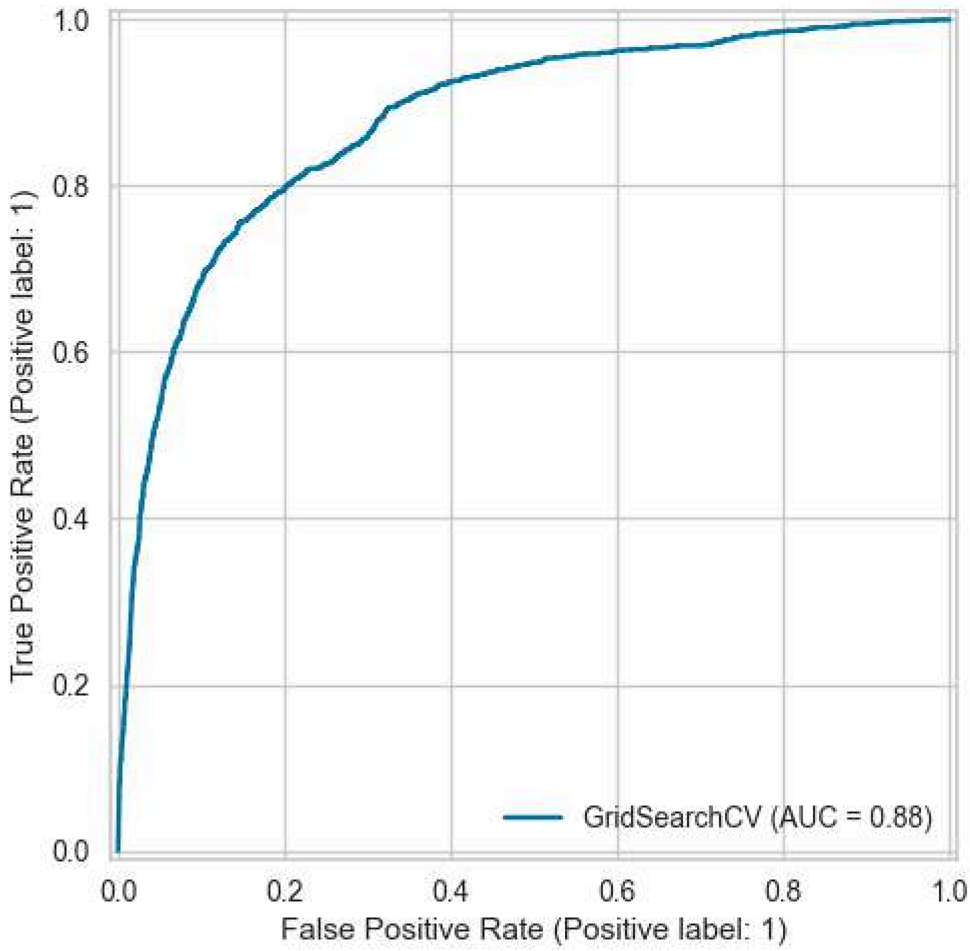
Out[114… 0.731964885295869

Precision Recall Curve and Roc Curve Display

In [52]:

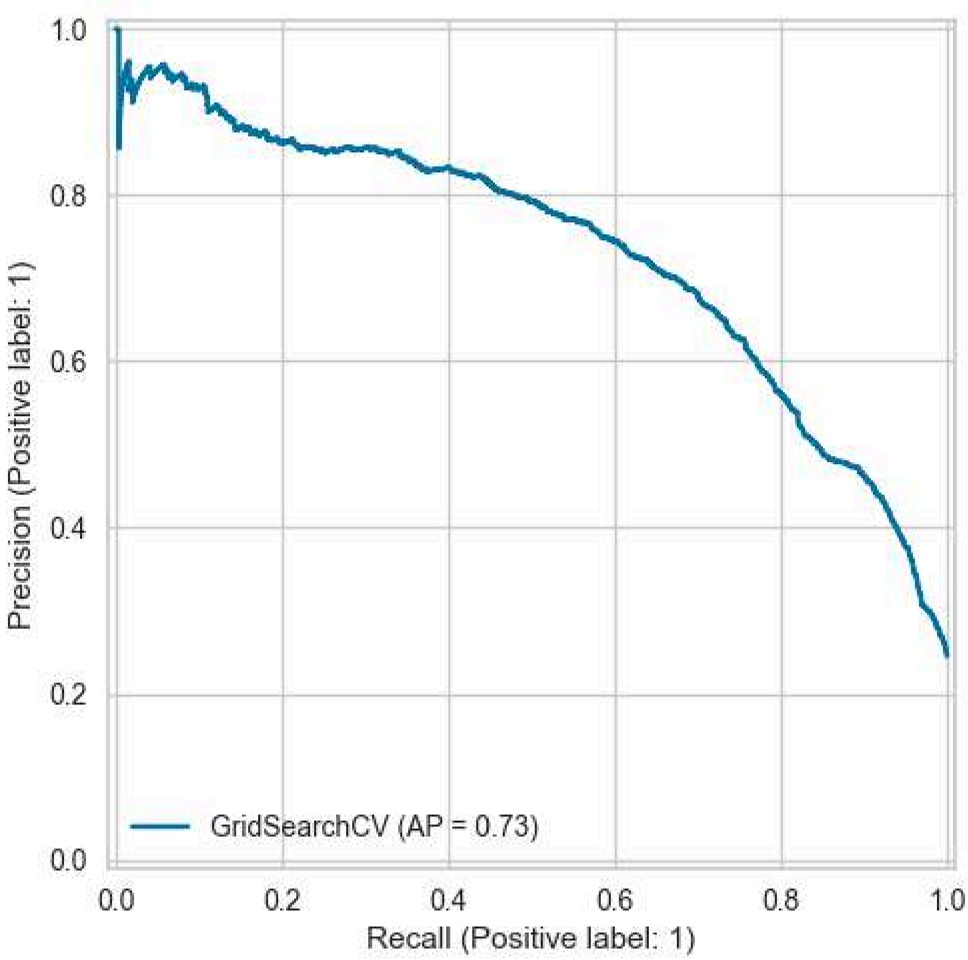
```
RocCurveDisplay.from_estimator(svm_model_grid, X_test, y_test);
```

Out[52]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2b5463ec710>



In [53]:
```
PrecisionRecallDisplay.from_estimator(svm_model_grid, X_test, y_test);
```

Out[53]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x2b54
625b950>
```



Compare Models Performance

In [115…

```
compare = pd.DataFrame({"Model": ["Logistic Regression", "KNN", "SVM"],
                        "F1": [log_f1, knn_f1, svm_f1 ],
                        "Recall": [log_recall, knn_recall, svm_recall ],
                        "ROC_AUC": [log_auc, knn_auc, svm_auc],
                         "PRC"   : [log_prc, knn_prc, svm_prc]})
def labels(ax):
    for p in ax.patches:
        width = p.get_width()                           # get bar length
        ax.text(width,                                  # set the text at 1 unit r
                p.get_y() + p.get_height() / 2,         # get Y coordinate + X coo
                '{:1.3f}'.format(width),                # set variable to display,
                ha = 'left',                            # horizontal alignment
                va = 'center')                          # vertical alignment
plt.figure(figsize=(14,12))

plt.subplot(411)
compare = compare.sort_values(by="F1", ascending=False)
ax=sns.barplot(x="F1", y="Model", data=compare, palette="magma")
labels(ax)

plt.subplot(412)
compare = compare.sort_values(by="Recall", ascending=False)
ax=sns.barplot(x="Recall", y="Model", data=compare, palette="magma")
labels(ax)
```

```
plt.subplot(413)
compare = compare.sort_values(by="ROC_AUC", ascending=False)
ax=sns.barplot(x="ROC_AUC", y="Model", data=compare, palette="magma")
labels(ax)

plt.subplot(414)
compare = compare.sort_values(by="PRC", ascending=False)
ax=sns.barplot(x="PRC", y="Model", data=compare, palette="magma")
labels(ax)

plt.show()
```

Final Model and Model Deployment

In [118…

```
operations = [("transformer", column_trans), ("logistic", LogisticRegression(ra

log_model = Pipeline(steps=operations)

param_grid = [
    {
        "logistic__penalty" : ['l1'],
        "logistic__C" : [0.03],
        "logistic__class_weight": ["balanced"] ,
        "logistic__solver": ['saga'],
        "logistic__max_iter": [1000]
    }
]
cv = StratifiedKFold(n_splits = 10)
```

```python
final_pipe_model = GridSearchCV(estimator=log_model,
                                param_grid=param_grid,
                                cv=cv,
                                scoring = "f1",
                                n_jobs = -1,
                                return_train_score=True).fit(X, y)
```

In [119…

```python
import pickle
pickle.dump(final_model, open("final_pipe_model", "wb"))
```

In [120…

```python
new_model = pickle.load(open("final_pipe_model", "rb"))
new_model
```

Out[120…

```
GridSearchCV
best_estimator_: Pipeline
    transformer: ColumnTransformer
        onehotencoder      ordinalencoder      remainder
        OneHotEncoder      OrdinalEncoder      StandardScaler
    LogisticRegression
```

Prediction

In [126…

```python
my_dict= {
    'age': [44.0, 32.0, 30.0],
    'workclass': ['Federal-gov', 'Private', 'Self-emp-not-inc'],
    'education': ['Bachelors', 'Bachelors', 'Some-college'],
    'education.num': [13.0, 13.0, 10.0],
    'marital.status': ['Widowed', 'Married', 'NotMarried'],
    'occupation': ['Tech-support', 'Sales', 'Sales'],
    'relationship': ['Not-in-family', 'Husband', 'Other-relative'],
    'race': ['White', 'White', 'Others'],
    'sex': ['Male', 'Male', 'Male'],
    'hours.per.week': [40.0, 40.0, 40.0],
    'native.country': ['United-States', 'United-States', 'Other'],
    'capital_diff': ['Low', 'Low', 'Low']
}
```

In [128…

```python
sample = pd.DataFrame(my_dict)
sample
```

Out[128…

	age	workclass	education	education.num	marital.status	occupation	relationship
0	44.0	Federal-gov	Bachelors	13.0	Widowed	Tech-support	Not-in-family
1	32.0	Private	Bachelors	13.0	Married	Sales	Husband
2	30.0	Self-emp-not-inc	Some-college	10.0	NotMarried	Sales	Other-relative

In [129…
```
new_model.predict(sample)
```
Out[129…
```
array([1, 1, 0], dtype=int64)
```
In [130…
```
new_model.decision_function(sample)
```
Out[130…
```
array([ 0.27369739,  1.16079392, -2.56091314])
```

📂 Conclusion

In [ ]:
```
#   Logistic grid      recall: 83,  f1 :  0.68   prc=0.75
```

- In an unbalanced dataset, F1-Score and Recall metrics are indeed very important. These metrics play a critical role in evaluating model performance in unbalanced datasets, as they measure the model's ability to correctly predict the minority class.

When prioritizing F1-Score and Recall:

- The Logistic Regression model stands out with a Recall of 0.83 and an F1-Score of 0.68. This model demonstrates balanced performance across the classes in the unbalanced dataset, effectively capturing the minority class while also performing well in overall classification.

- The KNN Model, although it performs well in terms of accuracy, lags behind Logistic Regression with a Recall of 0.59 and an F1-Score of 0.64. This indicates that the model is less effective at capturing the minority class in the unbalanced dataset.

- The SVM Model, despite excelling in accuracy, also falls behind Logistic Regression in these two metrics with a Recall of 0.60 and an F1-Score of 0.66. It is evident that SVM is not sufficiently successful in capturing the minority class.

Based on these results, I can say that the Logistic Regression model offers the best performance in terms of Recall and F1-Score for unbalanced datasets and should therefore be preferred. Especially in unbalanced datasets, it is critical that the model correctly identifies the minority class, making Logistic Regression the most suitable choice.

THANK YOU

If you want to be the first to be informed about new projects, please do not forget to follow us - by Fatma Nur AZMAN

Fatmanurazman.com | Linkedin | Github | Kaggle | Tableau