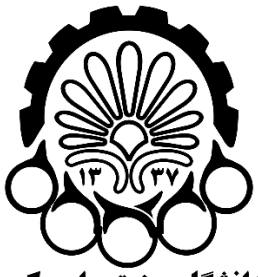


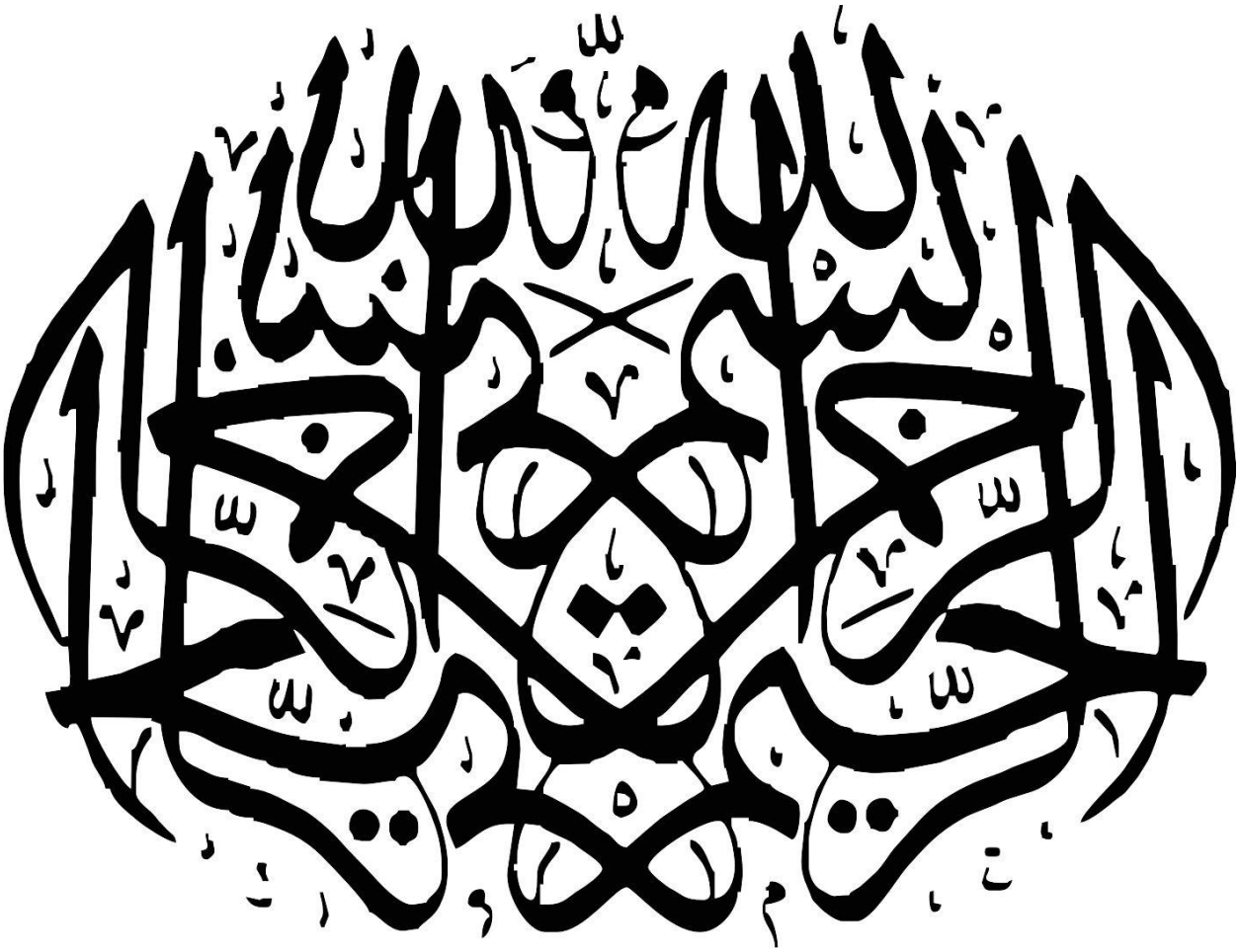
دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تکلیف اول درس یادگیری ماشین کاربردی

استاد درس: دکتر ناظرفرد

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: ۴۰۲۱۲۵۰۰۵



سید نیما محمودیان ۴۰۲۱۲۵۰۰۵

فهرست مطالب

فهرست شکل‌ها

۱-پاسخ سوال اول

۱-۱-لود کردن دیتاست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی داده‌ها

توجه شود که هرجا احتیاجی به ایپمورت کردن یک کتابخانه بود، برای حفظ نظم، ایمپورت در اولین بلوک کد انجام شده است شکل یک بلوک مربوط به وارد کردن کتابخانه‌ها را نمایش می‌دهد. برای لود کردن دیتاست، و نمایش ده ردیف رندوم از کتابخانه pandas استفاده می‌کنیم. با استفاده از کتابخانه numpy یک random seed ایجاد می‌کنیم. Random seed به مظور بازتولید نتایج مشابه تعریف می‌شود. شکل (۳)، خواندن دیتاست و نشان دادن ده ردیف رندوم را نمایش می‌دهد.

```
[1]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_auc_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

شکل ۱: وارد کردن کتابخانه‌های مورد نیاز

مشاهده می‌شود که دو ردیف به نام‌های Unnamed:0 و index در دیتاست وجود دارد که مقادیر آنها کاملاً منطبق بر شماره‌گذاری خودکاری است که pandas انجام می‌دهد. در نتیجه در قدم بعدی این دو ستون را با استفاده از متد ()drop حذف می‌کنیم. شکل (۲) نحوه‌ی حذف این دو ستون را نمایش می‌دهد.

```
Since Unnamed:0 and index are identical with the pandas indexing, we can drop them

df = df.drop(["Unnamed: 0", "index"], axis=1)
```

شکل ۲: حذف ستون‌های اضافه

در ادامه با استفاده از متد ()info نگاهی کلی به ستون‌های باقی‌مانده می‌اندازیم. سپس با استفاده از method chaining نمایش داده شده در شکل (۴) بررسی می‌کنیم که در هر ستون در مجموع چه تعداد مقادیر تعریف نشده وجود دارد.

با توجه به اطلاعات نمایش داده شده توسط متد ()info درمی‌یابیم که برخی از ستون‌ها دارای دیتاتایپ object هستند. شکل (۵) اطلاعات نمایش داده شده توسط این متد را نمایش می‌دهد.

Importing the dataset

```
df= pd.read_csv("Dataset_01.csv")
```

Showing ten random rows

```
# Set random seed for reproducibility
np.random.seed(42)

random_rows = df.sample(n=10)
random_rows
```

	Unnamed: 0	index	age	job_category	edu	edu_level	marriage	job	rel	race	sex	gain	loss	hours-per-week	nationality	salary
7762	7762	7762	56	Private	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	0	40	United-States	<=50K
23881	23881	23881	25	Private	HS-grad	9	Married-civ-spouse	Transport-moving	Own-child	Other	Male	0	0	40	United-States	<=50K
30507	30507	30507	43	Private	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	14344	0	40	United-States	>50K
28911	28911	28911	32	Private	HS-grad	9	Married-civ-spouse	Transport-moving	Husband	White	Male	0	0	40	United-States	<=50K
19484	19484	19484	39	Private	Bachelors	13	Married-civ-spouse	Sales	Wife	White	Female	0	0	30	United-States	<=50K
43031	43031	43031	20	Private	HS-grad	9	Never-married	Adm-clerical	Unmarried	White	Female	0	0	40	Germany	<=50K
28188	28188	28188	54	Private	HS-grad	9	Divorced	Transport-moving	Not-in-family	White	Male	0	0	45	United-States	<=50K
12761	12761	12761	25	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	White	Female	0	1887	40	United-States	>50K
40834	40834	40834	30	Private	HS-grad	9	Never-married	Machine-op-inspct	Not-in-family	White	Male	0	0	50	Dominican-Republic	<=50K
27875	27875	27875	33	Private	Some-college	10	Never-married	Adm-clerical	Own-child	White	Female	0	0	40	United-States	<=50K

شکل ۳: خواندن دیتاست و نمایش ده سطر تصادفی

Check if there are Nan values

```
df.isnull().sum()
```

```
age               0
job_category      0
edu               0
edu_level         0
marriage          0
job               0
rel               0
race              0
sex               0
gain              0
loss              0
hours-per-week    0
nationality       0
salary            0
dtype: int64
```

شکل ۴: بررسی مجموع مقادیر ناموجود در هر ستون

پس در قدم بعدی، ستون‌هایی که دارای تایپ object هستند را با استفاده از یک حلقه و متد ()astype به string تبدیل می‌کنیم.

در ادامه بررسی می‌کنیم که ستون‌هایی که مقدار دسته‌ای دارند، هر کدام شامل چه دسته‌هایی می‌شوند و از هر دسته چه تعداد داده در دیتاست موجود است. شکل (۶) نحوه انجام این کار را با استفاده از یک حلقه و متد ()value_counts را نمایش می‌دهد. در این حلقه، ستون‌ها یکی یکی فراخوانی می‌شوند و مقادیر هر دسته شمرده نمایش داده می‌شوند. توجه شود که در شکل (۶) بخشی از داده‌ها به نمایش درآمده‌اند.

با بررسی خروجی این متد به چند مورد مهم می‌رسیم.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   job_category    48842 non-null  object
 2   edu             48842 non-null  object
 3   edu_level       48842 non-null  int64
 4   marriage        48842 non-null  object
 5   job             48842 non-null  object
 6   rel             48842 non-null  object
 7   race            48842 non-null  object
 8   sex             48842 non-null  object
 9   gain            48842 non-null  int64
 10  loss            48842 non-null  int64
 11  hours-per-week  48842 non-null  int64
 12  nationality     48842 non-null  object
 13  salary          48842 non-null  object
dtypes: int64(5), object(9)
memory usage: 5.2+ MB
```

شکل ۵: اطلاعات نمایش داده شده توسط متد info()

```
for col in df.columns:
    if df[col].dtype == "string":
        print(f"{col}: {df[col].value_counts()}")
```

```
job_category: job_category
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
?                    2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: count, dtype: Int64
edu: edu
HS-grad         15784
Some-college    10878
Bachelors        8025
Masters          2657
Assoc-voc        2061
11th             1812
Assoc-acdm       1601
10th             1389
7th-8th           955
Prof-school       834
9th               756
12th              657
Doctorate         594
...
salary: salary
<=50K    37155
>50K     11687
Name: count, dtype: Int64
```

شکل ۶: تعداد مقادیر موجود در هر یک از متغیرهای دسته‌ای

- مقادیر ناموجود به جای Nan با علامت سوال مشخص شده‌اند
- دیتاست مربوط به یک مسئله دسته‌بندی دوتایی است

- در ستون nationality یک دسته به نام South وجود دارد. از آنجایی که این مقدار به کشور به خصوصی اشاره ندارد، مقادیر آن باید به نحوی مدیریت شوند
- کاردینالیتی دو دسته‌ی nationality و edu بالاست و این کار را برای آموزش مدل سخت می‌کند. در نتیجه باید کاردینالیتی این دو دسته نیز به نحوی مدیریت شود.

ابتدا به مدیریت کاردینالیتی داده‌ها می‌پردازیم. برای کاهش کاردینالیتی ستون nationality، همه‌ی کشورها به جز آمریکا را در دسته‌های بزرگتر مانند آمریکای شمالی، آمریکای جنوبی، اروپای شرقی، اروپای غربی و آسیا قرار می‌دهیم. از آنجایی که اکثر داده‌ها مربوط به کشور آمریکا هستند، برای کشور آمریکا هم یک دسته جدا از بقیه در نظر می‌گیریم. مقادیر علامت سوال را به همان شکل قبل نگهداری می‌کنیم. مقادیر South را نیز به علامت سوال تبدیل می‌کنیم.

با استفاده از یک دیکشنری این دسته بندی‌ها را به گونه‌ای ایجاد می‌کنیم که key ها اسامی کشورهای استفاده شده در دسته‌بندی باشند و valueها نام دسته‌ی جدید کشورها باشند. شکل (۷) نحوه انجام دسته‌بندی با استفاده از متد map() را نشان می‌دهد.

```python
# Mapping dictionary for broader categories
nationality_mapping = {
    'United-States': 'US',
    'Mexico': 'NorthAmerica',
    'Canada': 'NorthAmerica',
    'Puerto-Rico': 'NorthAmerica',
    'El-Salvador': 'NorthAmerica',
    'Guatemala': 'NorthAmerica',
    'Cuba': 'NorthAmerica',
    'Jamaica': 'NorthAmerica',
    'Haiti': 'NorthAmerica',
    'Dominican-Republic': 'NorthAmerica',
    'Outlying-US(Guam-USVI-etc)': 'US',
    'South': '?',
    'Columbia': 'SouthAmerica',
    'Peru': 'SouthAmerica',
    'Ecuador': 'SouthAmerica',
    'Nicaragua': 'SouthAmerica',
    'Trinadad&Tobago': 'SouthAmerica',
    'Honduras': 'SouthAmerica',
    'England': 'WesternEurope',
    'Germany': 'WesternEurope',
    'Italy': 'WesternEurope',
    'Poland': 'EasternEurope',
    'Hungary': 'EasternEurope',
    'Portugal': 'WesternEurope',
    'Ireland': 'WesternEurope',
    'France': 'WesternEurope',
    'Scotland': 'WesternEurope',
    'Yugoslavia': 'EasternEurope',
    'Greece': 'WesternEurope',
    'Holand-Netherlands': 'WesternEurope',
    'Philippines': 'Asia',
    'India': 'Asia',
    'China': 'Asia',
    'Vietnam': 'Asia',
    'Taiwan': 'Asia',
    'Iran': 'Asia',
    'Hong': 'Asia',
    'Cambodia': 'Asia',
    'Thailand': 'Asia',
    'Laos': 'Asia',
    'Japan': 'Asia',
    '?': '?'
}

# Map nationality to broader categories
df['nationality'] = df['nationality'].map(nationality_mapping)
```

شکل ۷: دسته‌بندی مجدد ستون nationality

سپس با همین روش، ستون edu را دسته‌بندی می‌کنیم. در این ستون، همه دسته‌ها را مانند قبل نگه می‌داریم ولی دسته‌های هفتم هشتم تا دوازدهم را به یک دسته به نام تحصیلات کمتر از دبیرستان تبدیل می‌کنیم. شکل هشت دسته‌بندی مربوط به این ستون را نمایش می‌دهد.

همچنین مشابه دو بخش قبلی، مقادیر ستون هدف را به مقادیر صفر و یک تبدیل می‌کنیم.

```python
# Mapping dictionary for broader categories
education_mapping = {
    'HS-grad': 'HS-grad',
    'Some-college': 'Some-college',
    'Bachelors': 'Bachelors',
    'Masters': 'Masters',
    'Assoc-voc': 'Assoc-voc',
    '11th': 'Less than High School',
    'Assoc-acdm': 'Assoc-acdm',
    '10th': 'Less than High School',
    '7th-8th': 'Less than High School',
    'Prof-school': 'Prof-school',
    '9th': 'Less than High School',
    '12th': 'Less than High School',
    'Doctorate': 'Doctorate'
}

# Map education level to broader categories
df['edu'] = df['edu'].map(education_mapping)

# Display the transformed column
df['edu'].value_counts()
```

[10]

```
edu
HS-grad                15784
Some-college           10878
Bachelors               8025
Less than High School   5569
Masters                 2657
Assoc-voc               2061
Assoc-acdm              1601
Prof-school              834
Doctorate                594
Name: count, dtype: int64
```

شکل ۸: دسته‌بندی مجدد ستون edu

۱-۲-مدیریت مقادیر گمشده

برای جایگذاری مقادیر گمشده دو روش را بررسی می‌کنیم. روش اول جایگذاری مقادیر با استفاده از مد، و روش دوم حذف ردیف‌هایی است که مقادیر گمشده دارند. سوالی که پیش می‌آید این است که کدام روش بهتر است؟ برای پاسخ به این سوال، ابتدا رویکردهای گفته شده را اجرا می‌کنیم. سپس روی دیتافریم ساخته شده، یک مدل جنگل تصادفی آموزش و تست می‌کنیم و نتیجه تست را با معیار مساحت زیر منحنی ROC بررسی می‌کنیم.

ابتدا مقادیر ؟ را با None جایگذاری می‌کنیم و سپس Noneها را با مقدار np.nan عوض می‌کنیم. دلیل این کار این است که متود Replace نمی‌تواند مقدار np.nan را استفاده کند. سپس از متد ()fillna استفاده می‌کنیم. این کلاس مقادیر None را شناسایی می‌کند و آنها را با np.nan جایگذاری می‌کند. دلیل اینکه بر استفاده از np.nan اصرار می‌شود این است که در ادامه برای جایگذاری این مقادیر از کلاس ()SimpleImputer استفاده می‌کنیم و این کلاس فقط می‌تواند مقادیر np.nan را شناسایی کند. شکل (۹) نحوه اجرای این کار را نمایش می‌دهد.

در ادامه دیتافریم را به دو قسمت آموزش و تست تقسیم می‌کنیم. سپس از آنجایی که از column transformerها برای تبدیل مقادیر به مقادیر موردنظرمان استفاده می‌کنیم، برای هر دسته از داده‌های عددی و دسته‌ای یک لیست جداگانه ایجاد می‌کنیم. کارکرد column transformerها به این صورت است که یک لیست از اسم ستون‌ها را دریافت می‌کند و تغییرات را ستون به ستون اعمال می‌کند. شکل (۱۰) تقسیم بندی به دو دسته آموزش و تست و جدا کردن نام ستون ها در دو لیست ستون‌های عددی و ستون‌های دسته‌ای را نمایش می‌دهد.

حال برای راحتی اجرای پیش پردازش، از کلاس Pipeline از کتابخانه scikitlearn استفاده می‌کنیم. این کلاس، مجموعه‌ای از اعمال را به شکل پیاپی بر روی دیتاست اجرا می‌کند. استفاده از پایپ‌لاین‌ها چند مزیت دارد:

- حجم کد نوشته شده را بسیار کاهش می‌دهد

```python
#replace "?" with Nan
df= df.replace({"?": None})
# Replace 'None' values with NaN
df.fillna(value=np.nan, inplace=True)
df
```

	age	job_category	edu	edu_level	marriage	job	rel	race	sex	gain	loss	hours-per-week	nationality	salary
0	25	Private	Less than High School	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	US	0
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	US	0
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	US	1
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	US	1
4	18	NaN	Some-college	10	Never-married	NaN	Own-child	White	Female	0	0	30	US	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	US	0
48838	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	US	1
48839	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	US	0
48840	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	US	0
48841	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	US	1

48842 rows × 14 columns

شکل ۹: جایگذاری ؟ با np.nan

```python
# Divide data into training and validation subsets
X_train, X_valid, y_train, y_valid = train_test_split(df.drop(["salary"], axis=1), df["salary"], train_size=0.6, test_size=0.4,random_state=0)
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
categorical_cols = [cname for cname in X_train.columns if X_train[cname].dtype in ["string",'object'] ]
# Select numerical columns
numerical_cols = [cname for cname in X_train.columns if X_train[cname].dtype in ['int64', 'float64']]
```

شکل ۱۰: train test split

- امکان بروز نشت داده (Data leakage) را کاهش می‌دهد.
- امکان امتحان چند رویکرد برای انجام یک کار را فراهم می‌کند.
- مدیریت متغیرها و دیتا تایپ‌ها را بسیار ساده می‌کند.

پایپ‌لاین مورد استفاده به دو قسمت تقسیم می‌شود: پیش‌پردازش داده‌های دسته‌ای، پیش‌پردازش داده‌های عددی، آموزش مدل و تست و ارزیابی مدل.

پیش‌پردازش داده‌های دسته‌ای خود شامل یک پایپ‌لاین کوچک‌تر است که شامل اجرای دو column transformer است. ترتیب وارد کردن ترنسفورمرها در شیِ پایپ‌لاین، نماینده ترتیب اجرا نیز می‌باشد. اولین ترنسفورمر SimpleImputer است که کار جایگزینی np.nan را با مد به عهده دارد. دومین ترنسفورمر روی هر ستون One hot encoding را اجرا می‌کند.

نحوه کارکرد این ترنسفورمرها به این صورت است که یک به یک ستون‌ها را دریافت می‌کنند، مقادیر خالی را با مد جایگذاری می‌کند و سپس ستون را به فرمت One hot تبدیل می‌کند. شکل (۱۱) پایپ‌لاین مربوط به پیش‌پردازش داده‌های ستونی را نمایش می‌دهد.

```python
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

شکل ۱۱: پایپ‌لاین پیش پردازش داده‌های دسته‌ای

برای پیش‌پردازش مقادیر عددی از کلاس MinMaxScaler() استفاده می‌کنیم. حال با استفاده از کلاس ColumnTransformer() یک شی به نام preprocessor ایجاد می‌کنیم. در این کلاس، لیستی از تاپل‌ها را به عنوان آرگومان پاس می‌دهیم به طوری که عضو اول تاپل، نام ترنسفورمر، عضو دوم عمل یا اعمالی است که باید روی ستون‌ها انجام شود و عضو سوم لیستی شامل نام ستون‌های هدف است که کارهای تعریف شده بر روی آنها اعمال می‌شود. شکل (۱۲) نحوه تعریف کلاس ColumnTransformer() را نمایش می‌دهد.

```
# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

شکل ۱۲: نحوه تعریف کلاس ColumnTransformer()

در قدم بعدی، مدل را تعریف می‌کنیم و پایپ‌لاین اصلی را ایجاد می‌کنیم که شامل دو مرحله‌ی پیش‌پردازش و آموزش است. شکل (۱۳) این بخش را نمایش می‌دهد.

```
model = RandomForestClassifier(n_estimators=100, random_state=0)


my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', model)
                             ])
```

شکل ۱۳: تعریف مدل و پایپ‌لاین اصلی

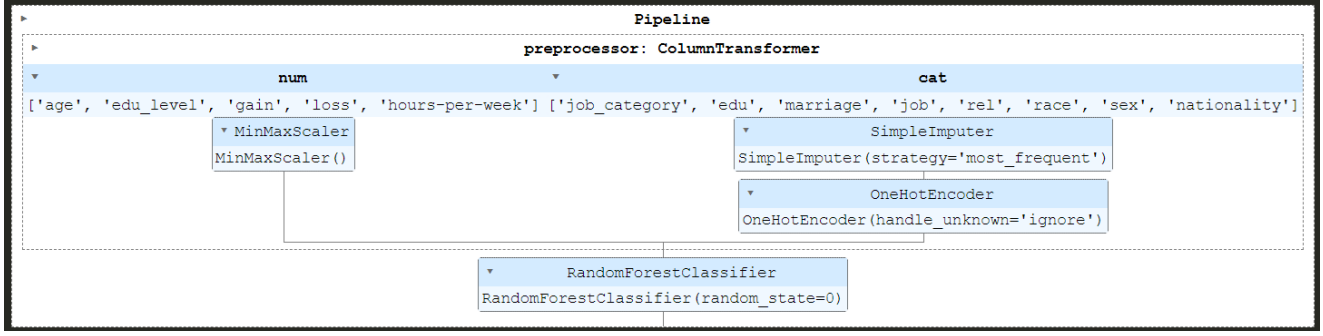
از آنجایی که این مدل صرفاً نقش یک معیار برای ارزیابی عملکرد روش‌ها را دارد، احتیاجی به fine tune کردن آن نیست چون که این مدل قرار نیست در ادامه کار پیش‌بینی داده‌های جدید را انجام دهد.

در قدم بعدی با استفاده از متد fit() و با پاس دادن داده‌های آموزشی به عنوان آرگومان، ابتدا داده‌ها را طبق توضیحات گفته شده پیش‌پردازش می‌کنیم و سپس مدل را آموزش می‌دهیم. شکل (۱۴) یک شمای کلی از پایپ‌لاین ایجاد شده را نمایش می‌دهد. همچنین لیست زیر عناوین num و cat نشان دهنده نام ستون‌هایی است که هر یک از پیش‌پردازش‌ها روی آنها انجام شده است.

در ادامه با استفاده از متد predict و پاس دادن داده‌های معیارسنجی، ابتدا این داده‌ها پیش‌پردازش می‌شوند و سپس مدل آموزش داده شده روی داده‌های آموزشی بر روی این داده‌ها معیارسنجی می‌شود. خروجی این دستور، ستونی از yهای پیش‌بینی شده است. با استفاده از دستور roc_auc_score و با پاس دادن لیبل‌های پیش‌بینی و لیبل‌های اصلی، مساحت زیر منحنی ROC محاسبه و نمایش داده می‌شود. شکل (۱۵) این بخش از کد را نمایش می‌دهد.

با استفاده از این روش مقدار AUC برابر با ۰.۷۶۸۹ می‌شود. حال به سراغ روش دوم که حذف سطرهای دارای مقدار ناموجود هستند می‌پردازیم.

```
# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)
```



شکل ۱۴: شمای کلی پایپ‌لاین

```
# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)

auc = roc_auc_score(y_valid, preds)
auc
```

```
0.7689806791903727
```

شکل ۱۵:معیارسنجی مدل

در این روش کاملاً مشابه به رویکرد پیش گرفته شده در بخش قبلی پیش می‌رویم با این تفاوت که متد SimpleImputer() را از پایپ‌لاین حذف می‌کنیم و در عوض قبل از تقسیم داده‌ها به داده‌های آموزشی و تست، سطرهای دارای مقادیر خالی را حذف می‌کنیم. بقیه روش مانند بخش قبل است. شکل (۱۶) نحوه حذف این سطرها با استفاده از متد dropna() را نمایش می‌دهد.

Deleting rows containing missing values

```
new_df= df.dropna()
```

شکل ۱۶: حذف سطرهای دارای مقادیر خالی

سپس مدلی با همان مشخصات قبلی روی این داده‌ها آموزش و معیار سنجی می‌شود. در این حالت میزان AUC برابر با ۰.۷۶۵۳ است که کمی کمتر از روش قبلی می‌باشد. در نتیجه برای پر کردن مقادیر خالی از مد استفاده می‌کنیم.

پس از انجام این کار نوع داده‌ی همه ستون ها به object تبدیل می‌شود که مشابه بخش قبلی نوع داده‌ها را مجدداً درست می‌کنیم.

۱-۳-مصور سازی داده‌ها

برای مصورسازی داده‌ها از دو کتابخانه matplotlib و seaborn استفاده می‌کنیم. ابتدا برای همه‌ی داده‌های دسته‌ای، نمودار میله‌ای رسم می‌کنیم. شکل (۱۷) نحوه انجام این کار با استفاده از کتابخانه matplotlib را نمایش می‌دهد.

```python
import matplotlib.pyplot as plt

# Filter categorical columns
categorical_columns = imputed_df.select_dtypes(include=['object', 'string'])

# Calculate the number of rows and columns needed for subplots
num_rows = 3
num_cols = 3

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 25))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot bar plots for each categorical column
for i, col in enumerate(categorical_columns):
    imputed_df[col].value_counts().plot(kind="bar", ax=axes[i])
    axes[i].set_title(col)

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plot
plt.show()
```

شکل ۱۷: کد رسم نمودار ستونی

نحوه عملکرد این کد در ادامه شرح داده می‌شود:

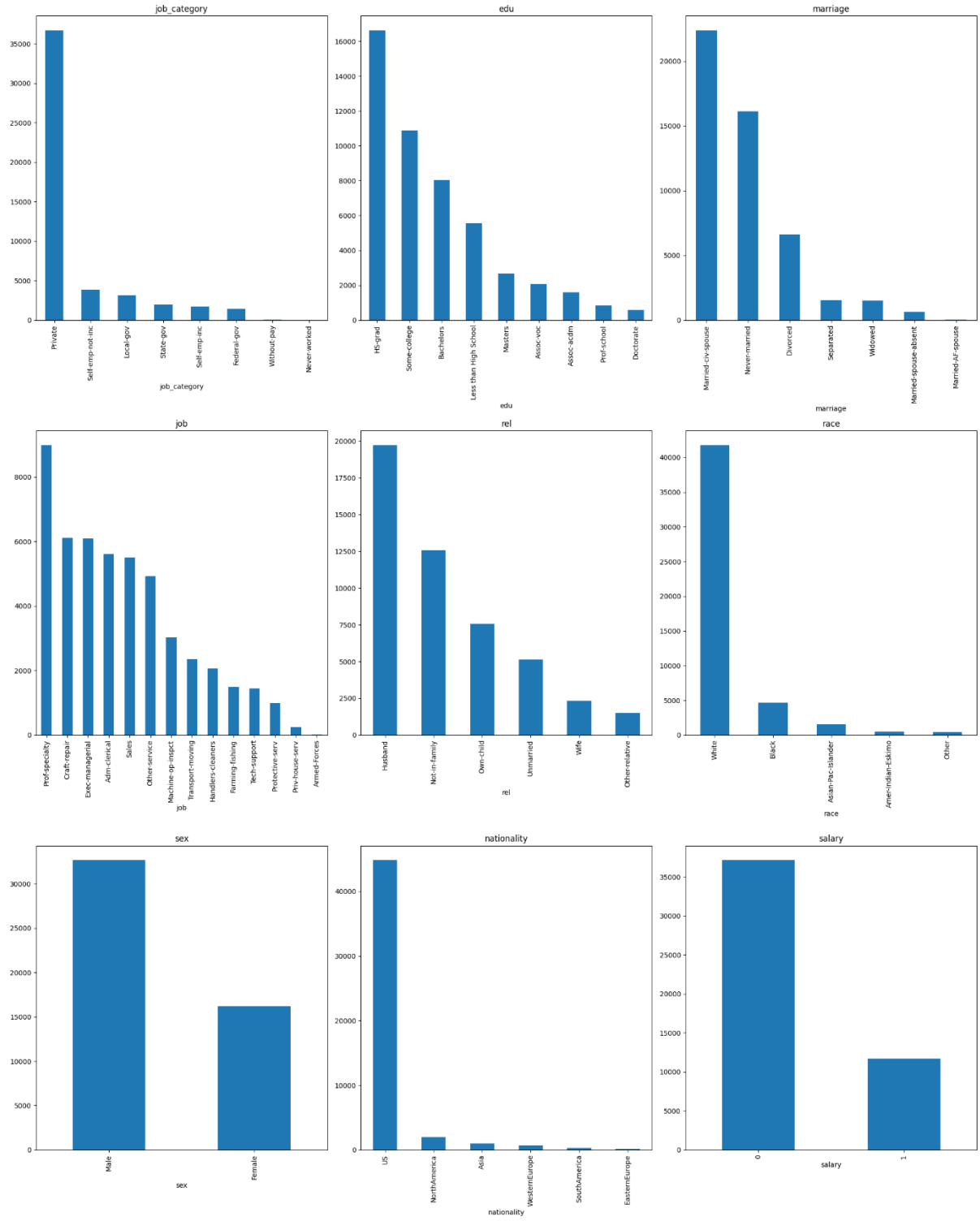
- categorical_columns = imputed_df.select_dtypes(include=['object', 'string']): ستون‌هایی را از imputed_df انتخاب می‌کند که دارای نوع داده 'object' یا 'string' هستند.
- num_rows = 3 و num_cols = 3: نمایش تعداد نمودارها در سطرها و ستون‌ها را تعیین می‌کند.
- fig, axes = plt.subplots(num_rows, num_cols, figsize=(20,25)): یک شکل و یک مجموعه از نمودارها با ابعاد num_rows توسط num_cols ایجاد می کند. پارامتر figsize اندازه کل شکل را بر حسب اینچ تنظیم می کند.
- سپس به ازای هر ستون در لیست ستون‌های دسته‌ای با استفاده از method chaining ابتدا تعداد مقادیر هر دسته محاسبه می‌شود و بر اساس آن یک نمودار میله‌ای رسم می‌شود.

شکل (۱۹) نیز خروجی این کد را نمایش می‌دهد.

سپس برای داده‌های عددی، نمودار جعبه‌ای و هسیتوگرام رسم می‌کنیم. شکل (۱۸) کد مربوط به نمودار جعبه‌ای را نمایش می‌دهد.

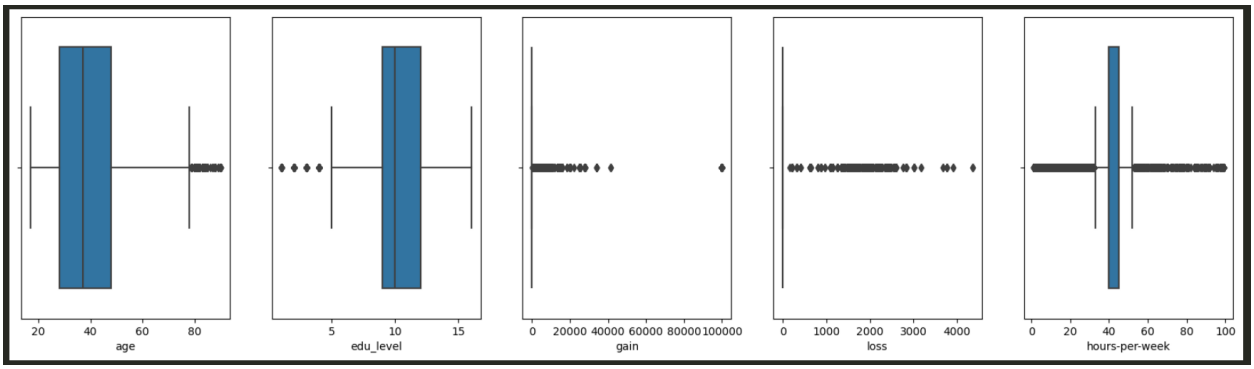
```python
import seaborn as sns
# Calculate the number of rows and columns needed for subplots
num_rows = 1
num_cols = 5

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 5))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot boxplots for each column
for i, col in enumerate(imputed_df.loc[:, numerical_cols]):
    sns.boxplot(x=imputed_df[col], ax=axes[i])

# Hide any remaining empty subplots
for i in range(len(imputed_df.columns), len(axes)):
    axes[i].axis('off')


# Show the plot
plt.show()
```

شکل ۱۸: کد مربوط به نمودار جعبه‌ای



شکل ۱۹: نمودارهای ستونی رسم شده از داده‌های دسته‌ای

عملکرد این کد مشابه کد قبلی است. ستون‌های عددی را جدا می‌کنیم، زیر نمودارها را ایجاد می‌کنیم و به ازای هر ستون عددی، نمودار جعبه‌ای رسم می‌کنیم. شکل (۲۰) نشان‌دهنده خروجی این کد است.



شکل ۲۰: نمودار جعبه‌ای از داده‌های عددی

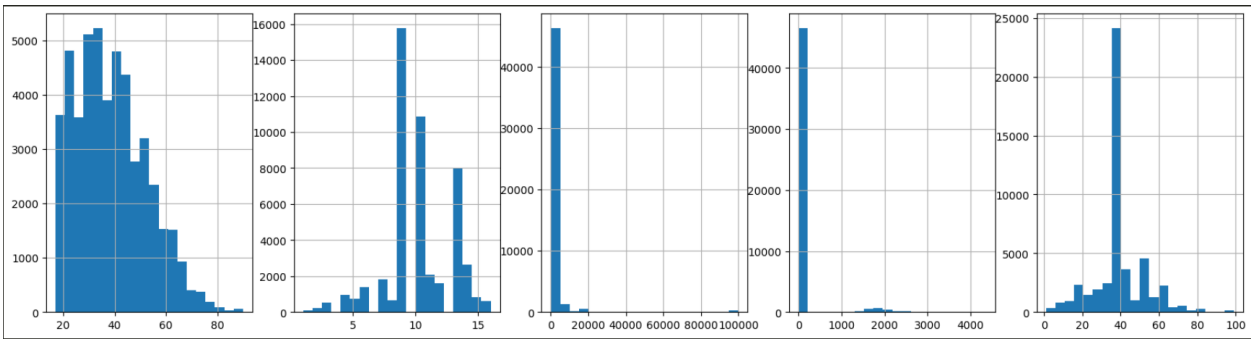
با توجه به شکل نمودارهای جعبه‌ای مشخص می‌شود که دیتاست حاوی داده‌های پرت است. پس از اتمام مصورسازی به مدیریت داده‌های پرت می‌پردازیم.

همچنین مشابه همین روش را برای رسم هیستوگرام پیاده‌سازی می‌کنیم. شکل (۲۱) کد مربوط به رسم هیستوگرام را نمایش می‌دهد.

```python
import seaborn as sns
# Calculate the number of rows and columns needed for subplots
num_rows = 1
num_cols = 5

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 5))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot boxplots for each column
for i, col in enumerate(imputed_df.loc[:, numerical_cols]):
    imputed_df[col].hist(ax=axes[i], bins=20)

# Hide any remaining empty subplots
for i in range(len(imputed_df.columns), len(axes)):
    axes[i].axis('off')


# Show the plot
plt.show()
```

شکل ۲۱: کد مربوط به هیستوگرام

همچنین شکل (۲۱) خروجی مربوط به این کد را نمایش می‌دهد.



شکل ۲۲: هیستوگرام‌های رسم شده

حال به رسم نمودارهای همبستگی می‌پردازیم. ابتدا نوع داده‌ی ستون salary را به float64 تغییر می‌دهیم تا بتوانیم ماتریس همبستگی و هیت‌مپ را رسم کنیم. برای انتخاب ستون‌های عددی از لیستی که بالاتر ایجاد کردیم استفاده می‌کنیم و برای انتخاب همه سطرها از ستون‌های عددی از iloc. استفاده می‌کنیم. سپس با استفاده از method chaining و استفاده از متد ()corr ماتریس همبستگی را محاسبه می‌کنیم. شکل (۲۳) نحوه انجام این کار را نمایش می‌دهد.

```
numerical_cols.append("salary")
```

```
imputed_df["salary"] = imputed_df["salary"].astype("float64")
```

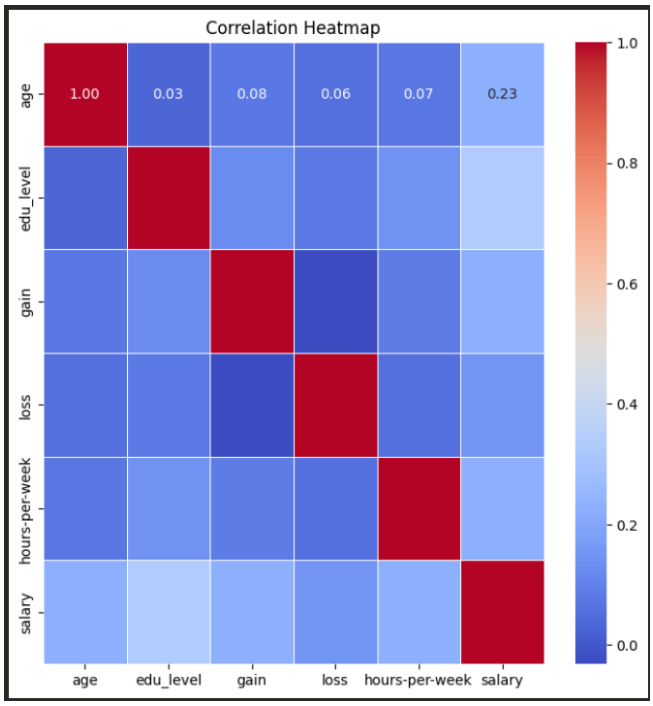
```
numerical_cols
```

```
['age', 'edu_level', 'gain', 'loss', 'hours-per-week', 'salary']
```

```
corr_matrix = imputed_df.loc[: , numerical_cols].corr()
plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

شکل ۲۳: نحوه انتخاب ستون‌های عددی و محاسبه ماتریس همبستگی

در نهایت هیت‌مپ مانند شکل (۲۴) رسم می‌شود.



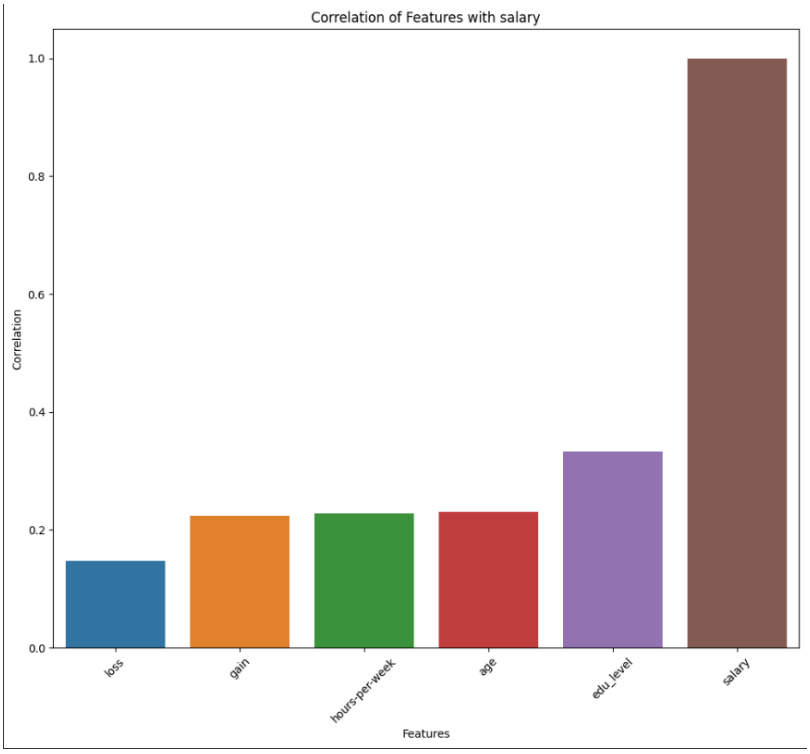
شکل ۲۴: هیت‌مپ رسم شده

همچنین بررسی همبستگی هر ستون با ستون هدف را نیز نمایش می‌دهیم. برای محاسبه این همبستگی پس از انتخاب همه ستون‌های عددی و همه‌ی سطرهای دیتافریم، با استفاده از متد ()corrwith، همبستگی این ستون‌ها را با ستون هدف محاسبه می‌کنیم. شکل (۲۵) نحوه کد کردن این بخش را نمایش می‌دهد.

```
# Compute correlation with label column
correlation = imputed_df.loc[: , numerical_cols].corrwith(imputed_df['salary'])
correlation = correlation.sort_values()
# Plot correlations
plt.figure(figsize=(12, 10))
sns.barplot(x=correlation.index, y=correlation.values)
plt.title('Correlation of Features with salary')
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.xticks(rotation=45)
plt.show()
```

شکل ۲۵: نحوه محاسبه همبستگی با ستون هدف

خروجی این کد در شکل (۲۶) نمایش داده شده است.



شکل ۲۶: نمودار میله‌ای همبستگی با ستون هدف

مشاهده می‌شود که ستون edu_level بیشترین همبستگی را با ستون هدف دارد.

حال به بررسی همبستگی ستون‌های دسته‌ای با ستون هدف می‌پردازیم. برای بررسی این همبستگی ابتدا باید ستون‌های دسته‌ای را One hot encode کنیم. برای one hot encode کردن این ستون‌ها از متد get_dummies() استفاده می‌کنیم. خروجی این متد یک دیتافریم دیگر است. با استفاده از متد concat() ستون هدف را در کنار این دیتافریم انکود شده قرار می‌دهیم. سپس همانند بخش قبلی هیت‌مپ را رسم می‌کنیم. شکل (۲۷) نحوه کد کردن این بخش را نمایش می‌دهد.

```
checking the correlation of categorical columns

categorical_df = pd.get_dummies(imputed_df.loc[: , categorical_cols])
categorical_df = pd.concat([categorical_df , imputed_df["salary"]])


corr_matrix = categorical_df.corr()
plt.figure(figsize=(16, 16))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

شکل ۲۷: نحوه محاسبه هیت‌مپ برای داده‌های دسته‌ای

شکل (۲۸) خروجی این کد را نمایش می‌دهد.

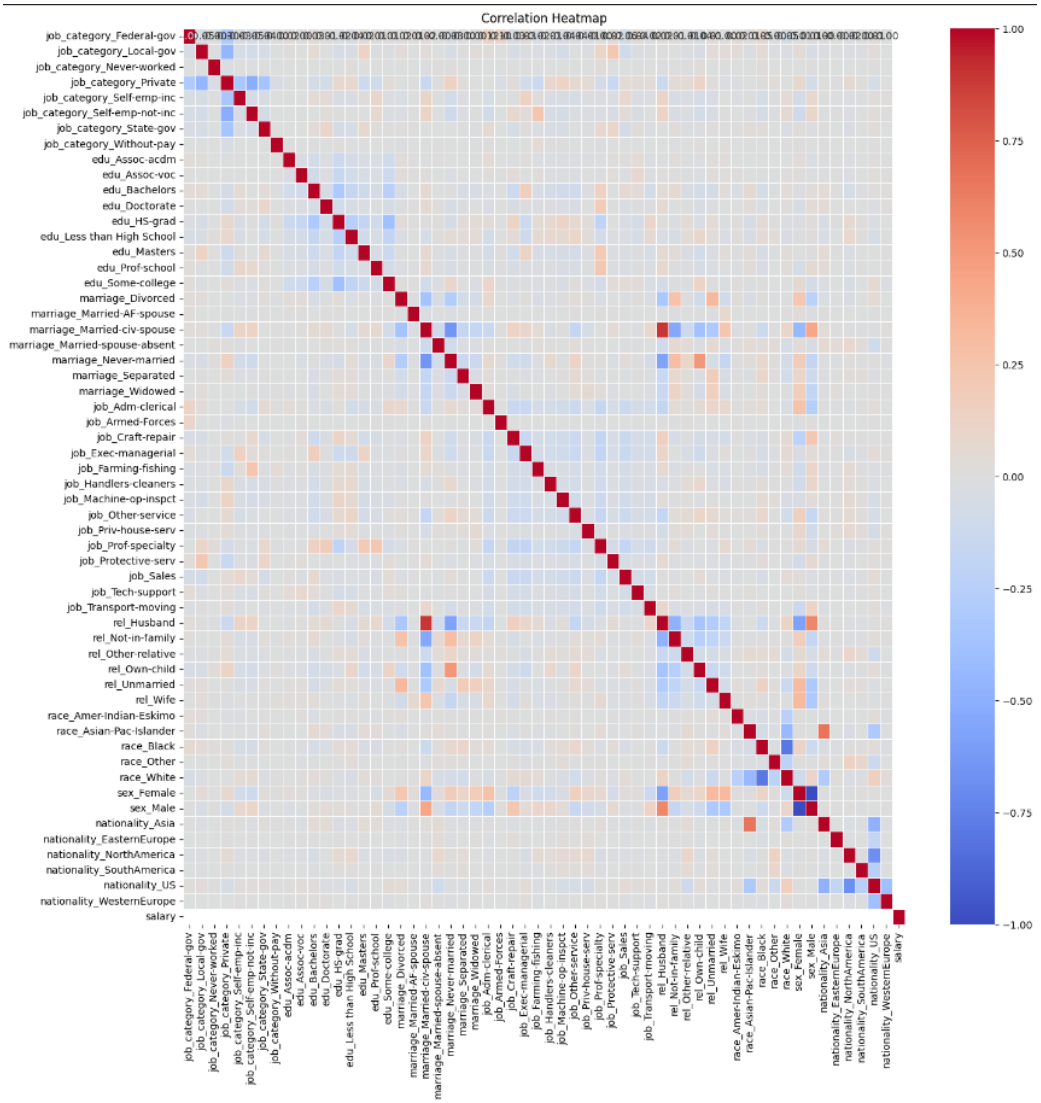
۱-۴-مدیریت داده‌های پرت

برای پیدا کردن داده‌های عددی پرت، از روش شش سیگما استفاده می‌کنیم. برای مدیریت این داده‌ها از دو روش استفاده می‌کنیم. اولین روش حذف آنها و روش دوم جایگزین کردن این داده‌ها با میانه است. روش ارزیابی و مقایسه این دو روش، همانند بخش مدیریت داده‌های ناموجود است. با استفاده از پایپ‌لاین این دو روش را جداگانه اعمال می کنیم و با آموزش دادن یک مدل ساده جنگل تصادفی و ارزیابی این مدل با AUC این دو روش را با هم مقایسه می‌کنیم.

ابتدا به بررسی روش حذف سطرهایی که داده پرت دارند می‌پردازیم. ابتدا این داده‌ها را با استفاده از کد نوشته شده در شکل (۲۸) حذف می‌کنیم. کارکرد این کد به این صورت است که برای هر ستون عددی میانگین و انحراف معیار را محاسبه می‌کنیم و سطرهایی که مقدار آنها از میانگین منهای سه سیگما کوچکتر و یا از میانگین به علاوه سه سیگما بزرگتر است را حذف می‌کنیم.

ابتدا دیتافریم را به دو دسته آموزشی و ارزیابی تقسیم می‌کنیم. سپس پایپ‌لاینی مشابه با پایپ‌لاین بخش حذف کردن سطرهای حاوی مقادیر گم‌شده ایجاد می‌کنیم. نحوه کارکرد این پایپ‌لاین دقیقاً مشابه به بخش‌های پیشین است. سپس مدل جنگل تصادفی را آموزش و ارزیابی می‌کنیم. در این روش، میزان AUC برابر با ۰.۷۴۸۴ به دست می‌آید.

روش دوم، جایگزینی داده‌های پرت با استفاده از میانه است. همانند بخش پیشین، داده‌های کمتر و بیشتر از آستانه را پیدا می‌کنیم و به جای حذف این داده‌ها، آنها را با میانه ستون متناظر جایگزین می‌کنیم. سپس با استفاده از پایپ‌لاین طراحی شده در بخش قبلی، مدل را آموزش و ارزیابی می‌کنیم. در این روش میزان AUC برابر با ۰.۷۵۵۹ به دست می‌آید.



شکل ۲۸: هیت‌مپ ایجاد شده از ستون‌های دسته‌ای

```
cleaned_df = imputed_df.copy()  # Create a copy of the original DataFrame

for col in numerical_cols:
    if col != "salary":
        lower_limit = imputed_df[col].mean() - 3 * imputed_df[col].std()
        upper_limit = imputed_df[col].mean() + 3 * imputed_df[col].std()

        # Filter rows to include only values within the threshold
        cleaned_df = cleaned_df[(cleaned_df[col] >= lower_limit) & (cleaned_df[col] <= upper_limit)]

# Reset index to ensure a continuous index after deleting rows
cleaned_df.reset_index(drop=True, inplace=True)
cleaned_df
```

	age	job_category	edu	edu_level	marriage	job	rel	race	sex	gain	loss	hours-per-week	nationality	salary
0	25.0	Private	Less than High School	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	US	0.0
1	38.0	Private	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	US	0.0
2	28.0	Local-gov	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	US	1.0
3	44.0	Private	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	US	1.0
4	18.0	Private	Some-college	10.0	Never-married	Prof-specialty	Own-child	White	Female	0.0	0.0	30.0	US	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45181	27.0	Private	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	US	0.0
45182	40.0	Private	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	US	1.0
45183	58.0	Private	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	US	0.0
45184	22.0	Private	HS-grad	9.0	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	US	0.0
45185	52.0	Self-emp-inc	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	US	1.0

45186 rows × 14 columns

شکل ۲۹: نحوه حذف سطرهایی که داده پرت دارند

در نتیجه برای مدیریت داده‌های پرت، از روش جایگزینی آنها با استفاده از میانه استفاده می‌کنیم.

۱-۵-مهندسی ویژگی‌ها

برای انتخاب ویژگی‌ها، از دو روش استفاده می‌کنیم: ۱-PCA و ۲-Mutual information همانند قسمت‌های پیش، برای مقایسه و ارزیابی این دو روش از پایپ‌لاین‌ها و یک مدل ساده جنگل تصادفی استفاده می‌کنیم که این مدل با AUC ارزیابی می‌شود. ابتدا به بررسی روش PCA می‌پردازیم.

ابتدا با استفاده از تابع get_dummies() دیتافریم تمیز شده را one hot encode می‌کنیم. از آنجایی که خروجی به شکل True و False است، برای تبدیل این مقادیر به صفر و یک، نوع ستون ها را به float64 تغییر می‌دهیم. شکل (۳۰) اجرای این دو کار را نمایش می‌دهد.

```
#First, let's onehot encode our categorical columns
dummy_df = pd.get_dummies(cleaned_df)
dummy_df
```

	age	edu_level	gain	loss	hours-per-week	salary	job_category_Federal-gov	job_category_Local-gov	job_category_Never-worked	job_category_Private
0	25.0	7.0	0.0	0.0	40.0	0.0	False	False	False	True
1	38.0	9.0	0.0	0.0	50.0	0.0	False	False	False	True
2	28.0	12.0	0.0	0.0	40.0	1.0	False	True	False	False
3	44.0	10.0	7688.0	0.0	40.0	1.0	False	False	False	True
4	18.0	10.0	0.0	0.0	30.0	0.0	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...
48837	27.0	12.0	0.0	0.0	38.0	0.0	False	False	False	True
48838	40.0	9.0	0.0	0.0	40.0	1.0	False	False	False	True
48839	58.0	9.0	0.0	0.0	40.0	0.0	False	False	False	True
48840	22.0	9.0	0.0	0.0	20.0	0.0	False	False	False	True
48841	52.0	9.0	15024.0	0.0	40.0	1.0	False	False	False	False

48842 rows × 63 columns

Converting Trues and Falses to 1 and 0

```
for col in dummy_df.columns:
    dummy_df[col] = dummy_df[col].astype("float64")
```

شکل ۳۰: one hot encoding

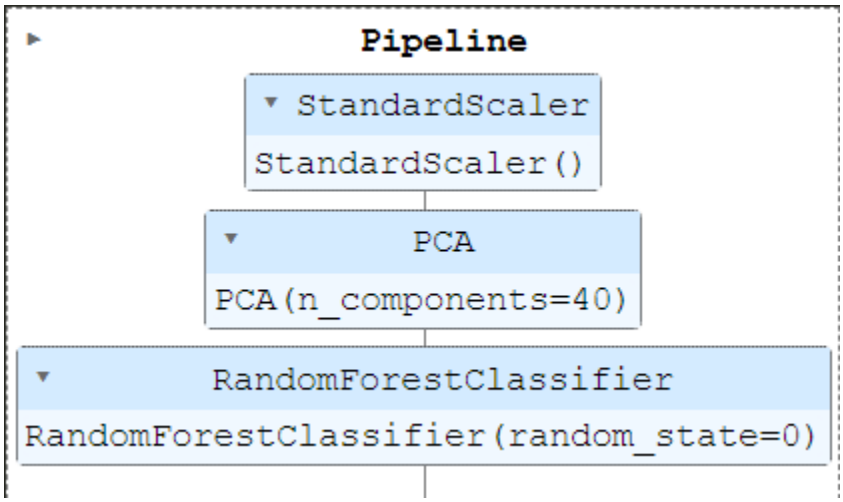
همچنین ستون هدف در وسط دیتافریم ایجاد شده قرار گرفته است. برای حفظ قرارداد که ستون هدف باید در ستون اول یا ستون آخر باشد، این ستون را به آخر دیتافریم انتقال می‌دهیم. شکل (۳۱) نحوه انجام این کار را نمایش می‌دهد.

در ادامه داده‌ها را به دو قسمت آموزش و ارزیابی تقسیم می‌کنیم، سپس یک پایپ‌لاین ایجاد می‌کنیم که در آن، ابتدا مقادیر را با استفاده از کلاس StandardScaler() استاندارد سازی می‌کنیم، سپس PCA را اجرا می‌کنیم و مدل را آموزش می‌دهیم. توجه فرمایید که پس از اجرای PCA، چهل PC اول را برای آموزش مدل انتخاب می‌کنیم. این مقدار با آزمون و خطا به دست آمده است. شکل (۳۲) یک شمای کلی از این پایپ‌لاین را نمایش می‌دهد. در نهایت داده‌های ارزیابی به مدل داده می‌شود و با مقایسه سطر هدف پیش‌بینی با سطر هدف ارزیابی، به مقدار ۰.۷۴۸۶ برای AUC می‌رسیم.

salary is in the middle of the DF. sending it to the last column for sake of convention

```
# Extract the 'salary' column
salary_column = dummy_df.pop('salary')
# Reinsert the 'salary' column at the end of the DataFrame
dummy_df['salary'] = salary_column
```

شکل ۳۱: انتقال ستون هدف به انتهای دیتافریم



شکل ۳۲: پایپ‌لاین ارزیابی روش PCA

حال به سراغ روش mutual information می‌رویم.

از آنجایی که روش mutual information یک کلاس نیست که اشیا آن متدهای ()fit و ()transform را داشته باشند، ابتدا به صورت جداگانه یک کلاس ایجاد می‌کنیم که استفاده از mutual information را در پایپ‌لاین امکان‌پذیر می‌کند. شکل ۳۳ نحوه ایجاد این کلاس و متدهای آن را نمایش می‌دهد.

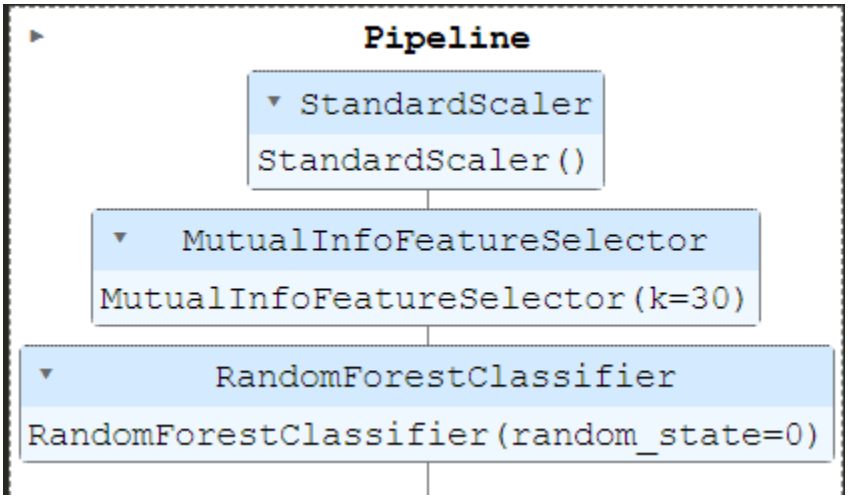
همچینین شکل (۳۴) شمای کلی پایپ‌لاین ایجاد شده را نمایش می‌دهد. این پایپ‌لاین ابتدا مقادیر را با استفاده از کلاس StandardScaler() استاندارد سازی می‌کند. سپس با استفاده از کلاس MutualInfoFeatureSelector() سی ویژگی که بیشترین mutual info را با سطر هدف دارد را انتخاب می‌کند و بر اساس آنها مدل جنگل تصادفی را آموزش می‌دهد. در این روش AUC برابر ۰.۷۴۸۱ می‌شود.

معیار ارزیابی این دو روش تقریباً با هم برابر است. هر دو روش هم پایه ریاضی قدرتمندی دارند. تفاوت مهمی که اینجا وجود دارد، این است که PCA ستون‌های ویژگی را کاملاً دگرگون می‌کند و آنها را به PC1,PC2 و... تبدیل می‌کند، ولی روش mutual information از همین ویژگی‌های موجود استفاده می‌کند. این رویکرد، تفسیرپذیری مدل‌هایی که قرار است روی دیتاست آموزش ببینند را بسیار بالاتر می‌برد. در نتیجه استفاده از Mutual information را بر PCA ترجیح می‌دهیم.

به انتهای سوال اول رسیدیم.

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

class MutualInfoFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, k=10):
        self.k = k

    def fit(self, X, y):
        self.selector = SelectKBest(mutual_info_classif, k=self.k)
        self.selector.fit(X, y)
        return self

    def transform(self, X):
        return self.selector.transform(X)

# Define the pipeline with Mutual Information, PCA, and RandomForest
pipeline = Pipeline([
    ('scaler', StandardScaler()),     # Standardize features
    ('mi_selector', MutualInfoFeatureSelector(k=30)),  # Mutual Information feature selection
    ('rf', RandomForestClassifier(n_estimators=100, random_state=0))  # RandomForest Classifier
])
```

شکل ۳۳: ایجاد کلاس و پایپ‌لاین برای mutual information



شکل ۳۴: شمای کلی پایپ‌لاین ایجاد شده برای mutual information

۲-پیش‌پردازش تصویر

ابتدا همانند سوال قبلی، تمامی کتابخانه‌هایی که در ادامه استفاده می‌شوند را در بلوک اول ایمپورت می‌کنیم. سپس یک رندوم سید برای بازتولید نتایج ایجاد می‌کنیم و سه عدد رندوم تولید می‌کنیم. در شکل (۳۵) تولید سه عدد تصادفی با استفاده از تابع randint() را نمایش می‌دهد.

سپس با استفاده از کتابخانه OpenCV و تابع imread سه عکس متناظر با اعداد تصادفی تولید شده را با استفاده از روش fstring فراخوانی می‌کنیم و در متغیرهایی با نام‌های img1، img2، و img3 ذخیره می‌کنیم. شکل (۳۶) یک نمونه کد برای لود کردن و ذخیره تصویر در متغیر را نمایش می‌دهد. همچنین برای اطمینان از درست لود شدن تصاویر، شرطی را قرار می‌دهیم که اگر مقدار لود شده برای عکس برابر None بود پیامی را نمایش دهد و در غیر اینصورت پیامی مبنی بر لود شدن موفقیت‌آمیز عکس را نمایش دهد.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

```
# Set the random seed
np.random.seed(42)
imgList= []
rand = np.random.randint(1 , 36 , size = 3)
rand
```

```
array([29, 15,  8])
```

شکل ۳۵: وارد کردن کتابخانه‌ها و تولید سه عدد تصادفی

Loading the images

```
# Read the image
img1 = cv.imread(f'Dataset_02/{rand[0]}.png')

# Check if the image is successfully loaded
if img1 is None:
    print("Error: Unable to read the image.")
else:
    print("Image successfully loaded.")
```

```
Image successfully loaded.
```

شکل ۳۶: لود کردن تصاویر و ذخیره آنها در متغیرها

سه بعد عکس‌ها عبارت‌اند از طول، عرض و تعداد کانال‌های رنگ. برای نمایش مقادیر این سه بعد، از اتریبیوت shape استفاده می‌کنیم. شکل (۳۷) نحوه نشان دادن سه بعد را نمایش می‌دهد.

```
# Get the dimensions of the image
height, width, channels = img1.shape

print(f"Height: {height}, Width: {width}, Channels: {channels}")
```

```
Height: 60, Width: 64, Channels: 3
```

شکل ۳۷: نمایش ابعاد عکس‌ها

برای نمایش عکس‌ها، ابتدا با استفاده از تابع cvtColor() آنها را از حالت BGR به RGB تبدیل می‌کنیم، سپس با استفاده از کتابخانه matplotlib این عکس‌ها را نمایش می‌دهیم. شکل (۳۸) کدِ نحوه نمایش این عکس‌ها را نشان می‌دهد.

شکل ۳۸: تبدیل عکس‌ها به rgb و نمایش آنها

۲-۱- gray scale کردن عکس‌ها

ابتدا به مزایا و معایب استفاده از عکس‌های رنگی و عکس‌های gray scale می‌پردازیم:

مزایای تصاویر رنگی:

- اطلاعات غنی‌تر: تصاویر رنگی علاوه بر روشنایی تصویر حاوی اطلاعاتی در مورد شدت رنگ ها نیز هستند و اطلاعات دقیق تری را برای تجزیه و تحلیل ارائه می دهند.
- تفسیر بصری بهتر: تصاویر رنگی اغلب تفسیر بصری بهتری را ارائه می دهند زیرا شباهت زیادی به آنچه چشم انسان می بیند، درک و تفسیر را برای انسان آسان تر می کند.
- استخراج ویژگی های پیشرفته: تصاویر رنگی می توانند ویژگی های اضافی را برای تجزیه و تحلیل، مانند اطلاعات بافت و شکل ارائه دهند، که می تواند برای کارهایی مانند تشخیص و تقسیم بندی اشیا مفید باشد.

معایب تصاویر رنگی:

- ابعاد بالاتر: تصاویر رنگی در مقایسه با تصاویر در مقیاس خاکستری ابعاد بالاتری دارند، زیرا دارای چند کانال رنگی هستند (به عنوان مثال قرمز، سبز، آبی) که می تواند پیچیدگی محاسباتی و نیازهای حافظه را افزایش دهد.
- افزایش نویز: تصاویر رنگی ممکن است به دلیل کانال های متعدد، نویز بیشتری داشته باشند، که می تواند بر دقت الگوریتم‌هایی که بر اطلاعات رنگی متکی هستند تأثیر بگذارد.
- پیش‌پردازش پیچیده: پیش پردازش تصاویر رنگی اغلب به تکنیک های پیچیده تری نیاز دارد، مانند تبدیل فضای رنگی و نرمال‌سازی کانال، که در مقایسه با تصاویر در مقیاس خاکستری می تواند از نظر محاسباتی فشرده‌تر و وقت‌گیر باشد.

مزایای تصاویر Grayscale:

- ابعاد کمتر: تصاویر در مقیاس خاکستری ابعاد کمتری در مقایسه با تصاویر رنگی دارند، زیرا فقط یک کانال نشان دهنده روشنایی دارند که منجر به کاهش پیچیدگی محاسباتی و نیازهای حافظه می شود.
- پیش‌پردازش ساده‌تر: پیش‌پردازش تصاویر در مقیاس خاکستری اغلب در مقایسه با تصاویر رنگی ساده‌تر است، زیرا آنها نیازی به تبدیل فضای رنگی و سایر مراحل پردازش مربوط به رنگ ندارند.
- کاهش نویز: تصاویر در مقیاس خاکستری ممکن است در مقایسه با تصاویر رنگی نویز کمتری از خود نشان دهند.

معایب تصاویر Grayscale:

از دست دادن اطلاعات: تصاویر در مقیاس خاکستری فقط اطلاعات روشنایی را می‌گیرند و اطلاعات رنگ را دور می‌اندازند، که ممکن است منجر به از دست دادن ویژگی‌های بالقوه مفید برای کارهای خاص شود.

تفسیر بصری محدود: تصاویر در مقیاس خاکستری ممکن است به اندازه تصاویر رنگی تفسیر بصری غنی ارائه نکنند، به ویژه برای کارهایی که رنگ نقش مهمی دارد، مانند تصویربرداری پزشکی یا سنجش از دور.

کاهش تمایز ویژگی: در برخی موارد، تصاویر در مقیاس خاکستری ممکن است در مقایسه با تصاویر رنگی، تبعیض ویژگی‌ها را کاهش دهند، زیرا فاقد ویژگی‌های مرتبط با رنگ هستند که می‌تواند برای برنامه‌های خاص مفید باشد.

حال در ادامه به نحوه gray scale کردن تصاویر می‌پردازیم. از تابع ()cvtColor استفاده می‌کنیم. آرگومان اول متغیری است که عکس در آن ذخیره شده و آرگومان دوم تغییری است که می‌خواهیم روی عکس اعمال شود که در این مورد برابر با cv.COLOR_BGR2GRAY است.

شکل (۳۹) نشان دهنده‌ی کد مربوط به gray scale کردن و نمایش دادن عکس است.

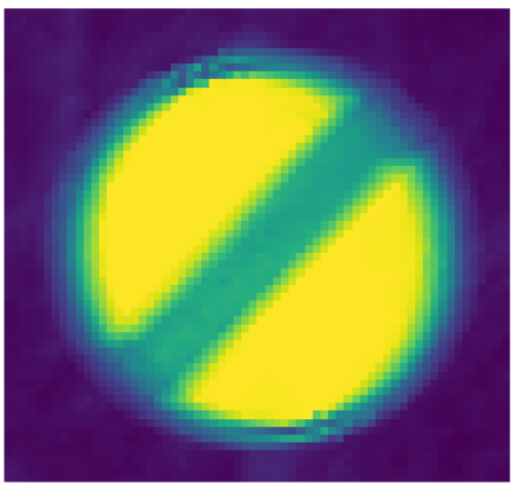
۲-۲-تنظیم روشنایی و کنتراست تصاویر

روشنایی و کنتراست تصویر نقش مهمی در تعیین کیفیت بصری و تفسیر یک تصویر دارند:

تنظیم روشنایی:

- تنظیم روشنایی می تواند تصویر را تیره یا روشن تر نشان دهد. افزایش روشنایی می‌تواند دید را در قسمت‌های تاریک‌تر تصویر افزایش دهد و جزئیات را قابل تشخیص‌تر کند. برعکس، کاهش روشنایی می‌تواند به کاهش نوردهی بیش از حد در مناطق روشن‌تر کمک کند و از از دست رفتن جزئیات جلوگیری کند.
- تنظیم مناسب روشنایی می تواند به بهبود ظاهر کلی تصویر کمک کند و آن را برای چشم دلپذیرتر کند. با این حال، تنظیمات بیش از حد روشنایی ممکن است منجر به از دست دادن جزئیات یا رنگ های شسته شده شود که بر دقت تفسیر تصویر تأثیر می گذارد.

```python
# Convert the image to grayscale
gray_img1 = cv.cvtColor(img1_rgb, cv.COLOR_BGR2GRAY)
# Display the grayscale image
plt.imshow(gray_img1)
plt.axis('off')  # Hide axis
plt.show()
```



شکل ۳۹: gary scale کردن عکس‌ها

تنظیم کنتراست:

- کنتراست به تفاوت در روشنایی یا رنگ بین روشن ترین و تاریک ترین قسمت های یک تصویر اشاره دارد. افزایش کنتراست تفاوت بین مناطق روشن و تاریک را افزایش می دهد و لبه ها و جزئیات را بیشتر مشخص می کند. کاهش کنتراست می تواند ظاهری نرم تر و کم صداتر ایجاد کند.
- تنظیم کنتراست بهینه می تواند به بهبود وضوح و وضوح جزئیات در یک تصویر کمک کند و قابلیت تفسیر آن را افزایش دهد. با این حال، تنظیمات کنتراست بیش از حد ممکن است منجر به از دست دادن جزئیات در هایلایت یا سایه ها شود و دقت تفسیر تصویر را کاهش دهد.

تنظیم روشنایی و کنتراست به چند دلیل می تواند به عنوان یک مرحله پیش پردازش در پردازش تصویر در نظر گرفته شود:

- افزایش دید: گاهی اوقات ممکن است تصاویر خیلی تاریک یا خیلی روشن به نظر برسند که تشخیص جزئیات را دشوار می کند. تنظیم روشنایی می‌تواند با روشن‌تر یا تیره‌تر کردن تصویر، بسته به نیازهای خاص، به بهبود دید کمک کند.
- بهبود کیفیت تصویر: تصاویر گرفته شده در شرایط نوری مختلف ممکن است سطوح متفاوتی از روشنایی و کنتراست داشته باشند. با تنظیم این پارامترها، می توانید ظاهر تصاویر را استاندارد کنید و به بهبود کیفیت کلی تصویر منجر شوید.
- ویژگی های برجسته: تنظیم کنتراست می تواند به برجسته کردن ویژگی های مهم در تصویر کمک کند. افزایش کنتراست، نواحی تاریک را تیره‌تر و نواحی روشن را روشن‌تر می‌کند، که می‌تواند به تأکید بر لبه‌ها و جزئیات کمک کند.
- نرمال سازی: در برخی موارد، تنظیم روشنایی و کنتراست می تواند توزیع شدت تصویر را عادی کند و استفاده از تکنیک های پردازش تصویر بعدی مانند تقسیم بندی، تشخیص اشیا یا طبقه بندی را آسان تر می کند.

- جلوگیری از نوردهی بیش از حد یا نوردهی کم: تصاویر گرفته شده در شرایط نوری شدید ممکن است از نوردهی بیش از حد (از دست دادن جزئیات در مناطق روشن) یا کم نور (از دست دادن جزئیات در مناطق تاریک) رنج ببرند. تنظیم روشنایی و کنتراست می تواند به اصلاح این مشکلات کمک کند تا اطمینان حاصل شود که جزئیات مهم حفظ می شوند.

برای تنظیم روشنایی و کنتراست، تابعی به صورت نمایش داده شده در شکل (۴۰) درست می‌کنیم.

شکل ۴۰: تابع تنظیم کنتراست و روشنایی

این تابع سه آرگومان دارد: عکسی که gray scale شده، ضریب آلفا و ضریب بتا. ضریب آلفا ضریب مقیاس برای تنظیم کنتراست است. مقدار ۱.۰ به معنای عدم تغییر است، مقادیر بیشتر از ۱.۰ کنتراست را افزایش می‌دهد و مقادیر کمتر از ۱.۰ کنتراست را کاهش می‌دهد. ضریب بتا مقداری است که برای تنظیم روشنایی به هر پیکسل اضافه می شود. مقادیر مثبت روشنایی را افزایش می دهد، در حالی که مقادیر منفی روشنایی را کاهش می دهد.

این تابع با استفاده از فرمول آلفا * img + بتا، تنظیم روشنایی و کنتراست را به تصویر ورودی اعمال می کند. این فرمول مقادیر پیکسل را با آلفا برای تنظیم کنتراست مقیاس می‌کند و سپس بتا را برای تنظیم روشنایی اضافه می‌کند. تابع np.clip برای اطمینان از اینکه مقادیر پیکسل در محدوده معتبر ۰ تا ۲۵۵ (برای یک تصویر ۸ بیتی در مقیاس خاکستری) هستند استفاده می شود. این امر از سرریز مقادیر پیکسل جلوگیری می کند. در نهایت، مقادیر پیکسل به نوع داده np.uint8 تبدیل می‌شوند، زیرا تصاویر در مقیاس خاکستری معمولاً از این نوع داده برای نمایش شدت پیکسل استفاده می‌کنند.

شکل (۴۱) نحوه استفاده از این تابع روی یکی از تصاویر را نمایش می‌دهد.

۲-۳-نرمال سازی تصاویر

نرمال‌سازی تصویر فرآیند تنظیم مقادیر پیکسل یک تصویر در مقیاس یا محدوده استاندارد است. هدف نرمال سازی، سازگاری بیشتر مقادیر پیکسل در تصاویر مختلف است که می تواند عملکرد مدل های یادگیری ماشین و سایر الگوریتم های پردازش تصویر را بهبود بخشد.

چالش هایی که ممکن است در صورت نرمال نبودن تصاویر بوجود آیند عبارتند از:

- محدوده شدت متناقض: تصاویر گرفته شده در شرایط نوری مختلف ممکن است دامنه شدت متفاوتی داشته باشند. بدون نرمال‌سازی، مقادیر پیکسل ممکن است به طور ناموزون در بین تصاویر مختلف توزیع شود، که مقایسه یا تجزیه و تحلیل موثر آنها را دشوار می‌کند.

شکل ۴۱: تنظیم روشنایی و کنتراست شکل اول

- حساسیت مدل: مدل‌های یادگیری ماشین، به‌ویژه مدل‌های یادگیری عمیق، اغلب به مقیاس ویژگی‌های ورودی حساس هستند. اگر مقادیر پیکسل تصاویر نرمال نباشد، می تواند منجر به همگرایی کُند در طول آموزش یا عملکرد ضعیف مدل شود.
- بیش‌برازش: توزیع مقادیر پیکسل ناسازگار در بین تصاویر می تواند منجر به بیش‌برازش، جایی که مدل یاد می‌گیرد به جای تعمیم به داده‌های جدید و نادیده، ویژگی‌های خاص داده‌های آموزشی را به خاطر بسپارد. نرمال‌سازی می‌تواند با اطمینان از اینکه مدل به جای مقادیر پیکسل خاص بر الگوهای اساسی در داده ها تمرکز می کند، به کاهش خطر بیش‌برازش کمک کند.
- دشواری در مقایسه: نرمال‌سازی تصاویر تضمین می کند که آنها دارای ویژگی های آماری مشابه هستند و مقایسه یا تجزیه و تحلیل آنها را آسان‌تر می کند. بدون عادی سازی، مقایسه بین تصاویر ممکن است گمراه کننده یا نادرست باشد.
- تقویت نویز: در برخی موارد، نرمال‌سازی می‌تواند به کاهش تأثیر نویز در تصاویر با مقیاس‌گذاری مقادیر پیکسل به محدوده کوچک‌تر کمک کند. بدون نرمال سازی، نویز ممکن است تقویت شود و استخراج اطلاعات معنی‌دار از تصاویر دشوارتر شود.

شکل (۴۲) تابعی را نمایش می‌دهد که یک تصویر Gray scale شده را دریافت می‌کند، مقادیر پیکسل‌ها را به ۲۵۵ تقسیم می‌کند و تصویر نرمال شده را برمی‌گرداند.

شکل (۴۳) یک نمونه استفاده از این تابع بر روی عکس‌ها و نتیجه آن را نمایش می‌دهد.

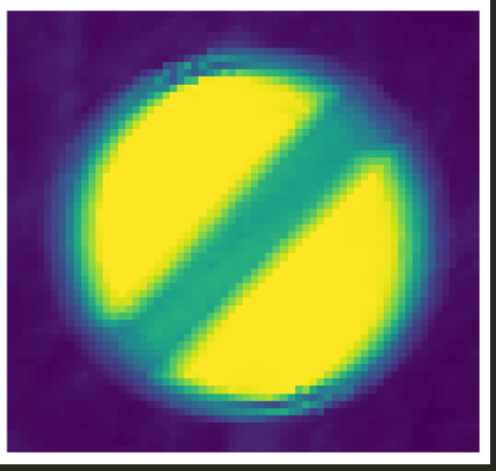
با انجام نرمال‌سازی، تمامی موارد اشاره شده در سوال دوم انجام شدند.

در قسمت بعدی به پیش‌پردازش متن می‌پردازیم.

```
Normalizing the images

def normalize_image(image):
    # Convert image to floating point format
    normalized_image = image.astype('float32')

    # Normalize pixel values to the range [0, 1]
    normalized_image /= 255.0

    return normalized_image
```

شکل ۴۲: تابع نرمال کننده عکس

```
plt.imshow(normalized_img1)
plt.axis('off')  # Hide axis
plt.show()
```



شکل ۴۳: نرمال کردن عکس با استفاده از تابع

۳-پیش‌پردازش متن

ابتدا همانند قسمت‌های پیشین، کتابخانه‌های مورد نیاز را در اولین بلوک وارد می‌کنیم. شکل (۴۴) این کتابخانه‌ها را نمایش می‌دهد. در قدم بعدی باید پیکره روزنامه همشهری را به شکل مناسب وارد محیط پایتون کنیم.

۳-۱-خواندن داده‌ها از پیکره همشهری

توجه فرمایید که اجرای مجدد کدهای این بخش به علت زمان طولانی پردازش، پیشنهاد نمی‌شود.

همچنین تمامی فایل‌های اکسل ایجاد شده به همراه نوت‌بوک‌ها، عکس‌ها و دیتافریم‌های ایجاد شده در لینک گوگل درایو حاضر در پیوست قرار داده شده است. به علت حجم بالای این فایل‌ها، امکان آپلود آنها در فایل تکالیف وجود ندارد.

برای خواندن داده‌ها ابتدا نگاهی به فایل تکست می‌اندازیم. مشاهده می‌شود که این فایل یک الگوی مشخص دارد. این الگو به این صورت است که مقدار مربوط به سه فیلد اول، یعنی DID، Date، Cat پس از یک فاصله tab وارد شده است. سپس متن اصلی محتوا آمده. در نهایت پس از اتمام محتوا، خبر بعدی با یک فاصله سه خطی از آخرین خطی که محتوای خبر قبلی در آن قرار دارد شروع می‌شود.

این شکل از الگو را می‌توان با کمی تلاش با استفاده از رجکس جداسازی و در یک دیتافریم قرار داد.

```python
import pandas as pd
import re
import string
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from hazm import Normalizer, word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
```

شکل ۴۴: کتابخانه‌های استفاده شده در سوال سوم

ایده کلی، خواندن این الگو در فایل تکست، ایجاد یک دیکشنری متشکل از چهارلیست با استفاده از چهار کلید DID، Date، Cat، و Content جداسازی هر یک از بخش‌های DID، Date، Cat، و Content و قرار دادن آنها در لیست متناظر به هر کدام است. شکل (۴۵) تابعی که مسئول انجام این کار است را نمایش می‌دهد.

```python
def parse_text_blocks(text):
    # Define the regex pattern to match each block of text
    pattern = r'\.DID\s+(.*?)\n\.Date\s+(.*?)\n\.Cat\s+(.*?)\n(.*?)\n\n\n'

    # Find all matches of the pattern in the text
    matches = re.findall(pattern, text, re.DOTALL)

    # Create a dictionary to store parsed data
    parsed_data = {'DID': [], 'Date': [], 'Cat': [], 'Content': []}

    # Iterate over matches and populate the dictionary
    for match in matches:
        did, date, cat, content = match
        parsed_data['DID'].append(did.strip())
        parsed_data['Date'].append(date.strip())
        parsed_data['Cat'].append(cat.strip())
        parsed_data['Content'].append(content.strip())

    return parsed_data
```

شکل ۴۵: جداسازی رکوردها و قرار دادن آنها در چهار فیلد

در این تابع ابتدا یک الگوی رجکسی تعریف شده که این همان الگویی است که بالاتر تعریف شد، متد findall() این الگو را به همراه متن دریافت می‌کند و نتایج را در متغیر matches ذخیره می‌کند. سپس به ازای هر یک از اعضای حاضر در متغیر matches، مقادیر را به لیست متناظر با هر کلید از دیکشنری تخصیص می‌دهد. در نهایت این تابع دیکشنری ایجاد شده را بر می‌گرداند.

مزیت استفاده از این رویکرد، خوکار بودن عمل پیداکردن الگو و تخصیص به دیکشنری است. همچنین دیکشنری‌هایی که value آنها به صورت لیست است به سادگی توسط pandas به دیتافریم تبدیل می‌شوند.

همچنین برای دیباگینگ راحت‌تر، به جای اینکه کل فایل تکست به این تابع پاس داده شود، ابتدا فایل توسط تابع split_file_into_chunks() به یازده قسمت تقسیم شده و هر قسمت به طور مجزا به تابع parse_text_blocks() پاس داده می‌شود. این تابع فایل تکست را می‌خواند، طول کلی آن را محاسبه می‌کند، آن را به یازده بخش تقسیم می‌کند و هر بخش را به شکل یک رشته در لیستی به نام chuck ذخیره می‌کند و در نهایت این لیست را بر می‌گرداند. شکل (۴۶) این تابع را نمایش می‌دهد.

The file it too large to extract all the data at once. so I split it into 11 parts and extract data of each part separately. It also makes debugging much easier

```python
def split_file_into_chunks(file_path, num_chunks):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    total_length = len(text)
    chunk_size = total_length // num_chunks

    # Split the text into chunks
    chunks = [text[i:i+chunk_size] for i in range(0, total_length, chunk_size)]

    return chunks
# Split the main file into chunks
num_chunks = 10
file_chunks = split_file_into_chunks(file_path, num_chunks)
```

شکل ۴۶: تابع تقسیم کننده فایل به یازده بخش

از آنجایی که انجام این فرایند طولانی است، ترجیح بر آن است که بخش‌هایی که تبدیل به دیتافریم شده‌اند را در فایل‌های اکسل جداگانه ذخیره کنیم. برای این منظور تابعی به نام export_to_excel() تعریف می‌کنیم که پس از ایجاد دیتافریم، در درون خود یک تابع دیگر به نام clean_string() را صدا می‌زند. کاربرد این تابع این است که تمامی حروفی که در مجموعه printable_chars است را حذف و به جای آنها یک فاصله قرار می‌دهد. این مجموعه شامل تمامی حروف فارسی و انگلیسی و حروفی از عربی است که در فایل استفاده شده‌اند. پس از پاکسازی سطر به سطر متن‌ها با استفاده از تابع apply()، می‌توان دیتافریم ایجاد شده را در یک فایل اکسل ذخیره کرد. نتیجه ذخیره فایل در قالب یک پیام نمایش داده خواهد شد. شکل (۴۷) دو تابع و مجموعه کاراکترها را نمایش می‌دهد.

```python
# Define a string containing all printable ASCII characters
printable_chars = set(string.printable + 'ئآابپتثجچحخدذرزژسشصضطظعغفقکگلمنوهییکۀئ')

def clean_string(s):
    # Replace any character not in the printable ASCII range with a space
    return ''.join(c if c in printable_chars else ' ' for c in s)

def export_to_excel(parsed_data, file_name):
    # Create a DataFrame from parsed data
    df = pd.DataFrame(parsed_data)

    # Remove any illegal characters from the DataFrame
    df = df.applymap(clean_string)

    try:
        # Export DataFrame to Excel
        df.to_excel(file_name, index=False)
        print(f"Data successfully exported to {file_name}")
    except Exception as e:
        print(f"An error occurred while exporting data to {file_name}: {e}")
```

شکل ۴۷: تمیز کردن متن‌ها و ذخیره آنها در اکسل

حال همه فایل‌های اکسل ایجاد شده را با استفاده از list comprehension در یک لیست فراخوانی می‌کنیم (لیستی از دیتافریم‌ها ایجاد می‌کنیم.) سپس با تابع concat() همه دیتافریم‌های این لیست را در کنار هم قرار می‌دهیم و در متغیر df ذخیره می‌کنیم. شکل (۴۸) انجام این کار را نمایش می‌دهد.

```python
dfList = [pd.read_excel(f'parsed_data_part_{i}.xlsx') for i in range(0,11)]
```

```python
df = pd.concat(dfList, axis=0)
df
```

	DID	Date	Cat	Content
0	1S1	75\04\02	adabh	...نگاهي به\n جاودانگي در زندگي گروهي از طريق هنر
1	2S1	75\04\02	adabh	...نمايشگاه هنر در خدمت ديك\n رويدادهاي هنري جهان
2	3S1	75\04\02	adabh	...نمايشگاه\n :گالري گلستان\n بردیوار نگارخانه ها
3	4S1	75\04\02	ejtem	...مطالعه اي مقدماتي پيرامون\n بازي را جدي بگيريم
4	5S1	75\04\02	elmfa	...;اشاره\n ...تخته سياه و غباري كه ستره نمي شود
...	...	...	...	...
10686	60055S1	81\11\20	vrzsh	...گروه ورزشي: با ح\nگره هاي كور كشتي باز مي شود
10687	60055S2	81\11\20	vrzsh	... از ايران هر \n نماينده فدراسيون جهاني واليبال
10688	60055S3	81\11\20	vrzsh	...گر\n شكست نامداران تكواندودر پيكارهاي برتر ليگ
10689	60055S4	81\11\20	vrzsh	... ساخته مي\n ورزشگاه بزرگ دانشگاه آزاد در تهران
10690	60055S5	81\11\20	vrzsh	...گروه ورزشي: مج\n رئيس فدراسيون پزشكي انتخاب شد

165215 rows × 4 columns

شکل ۴۸: تبدیل فایل‌های اکسل به یک دیتافریم

۳-۲-پیش‌پردازش متن‌ها

پس از درست کردن دیتافریم، حال زمان آن است که متن‌ها را پیش پردازش کنیم. ابتدا به چند مورد کوچک می‌پردازیم. اولین مورد بررسی نحوه انکود شدن متن است. تابع شکل (۴۹) تلاش می‌کند متن‌ها را با روش‌های مختلف انکود و دیکود کند تا انکودینگ اصلی متن را پیدا کند.

```
check the encoding

# Try to decode using different encodings
encodings_to_try = [ 'utf-8','latin-1', 'utf-16', 'ascii']  # Add other encodings as needed
for encoding in encodings_to_try:
    try:
        decoded_content = df['Content'].apply(lambda x: x if isinstance(x, str) else str(x))
        decoded_content.apply(lambda x: x.encode(encoding).decode(encoding))
        print(f"Decoding successful with encoding: {encoding}")
        break
    except UnicodeDecodeError:
        print(f"Failed to decode with encoding: {encoding}")

Decoding successful with encoding: utf-8
```

شکل ۴۹: پیدا کردن نحوه انکودینگ متن

مشاهده می‌شود که متن قابلیت انکود و دیکود با روش‌های utf-8 و utf-16 را دارد.

در ادامه کاراکترهای ناخواسته را حذف یا جایگزین می‌کنیم. در ستون Content تمامی کاراکترهای \n باید حذف شوند و در ستون Date تمامی کاراکترهای \\ باید با – جایگزین شوند. این کار با استفاده از متد str.replace امکان‌پذیر است. در نهایت سه ستون به شکل مناسب type cast می‌شوند. شکل (۵۰) نحوه انجام این سه کار را نمایش می‌دهد.

```
remove "\n"s

# Remove the "\n" characters from the text column
df['Content'] = df['Content'].str.replace('\n', '')

replace '\' with '-'

df['Date'] = df['Date'].str.replace('\\', '-')

type casting

df["DID"] = df["DID"].astype("str")
df['Cat'] = df['Cat'].astype('category')
df["Content"] = df["Content"].astype("str")
```

شکل ۵۰: حذف و جایگذینی کاراکترها و type casting

حال به پیش‌پردازش کاراکترها می‌پردازیم.

در اولین قدم، علائم نگارشی را حذف می‌کنیم. این کار با استفاده از تابع str.maketrans() صورت می‌گیرد این تابع مجموعه‌ای از علائم نگارشی را دریافت می‌کند و هنگامی که در متن به این علائم می‌رسد، آنها را با یک فضای خالی جایگزین می‌کند. این تابع در یک متد apply() قرار می‌گیرد و تغییرات را سطر به سطر روی ستون Content پیاده می‌کند.

Removing the punctuations

```
# Function to remove punctuation from text
def remove_punctuation(text):
    # Define a translation table with all punctuation characters mapped to None
    translator = str.maketrans('', '', string.punctuation)
    # Remove punctuation using the translation table
    return text.translate(translator)

# Apply the function to the "Content" column
df['Content'] = df['Content'].apply(remove_punctuation)
df
```

شکل ۵۱: حذف علائم نگارشی

در ادامه تمامی اعداد را با استفاده از رجکس حذف می‌کنیم. با استفاده از re.sub() تعریف می‌کنیم که هرکجا در متن به عدد رسیدیم، این عدد با یک فاصله تعویض شود. شکل (۵۲) نحوه انجام این کار را نمایش می‌دهد.

Removing the numbers

```
# Function to remove numbers from text using regular expressions
def remove_numbers(text):
    # Use regular expression to remove all numbers
    return re.sub(r'\d+', '', text)

# Apply the function to the "Content" column
df['Content'] = df['Content'].apply(remove_numbers)

# Display the DataFrame
df
```

شکل ۵۲: نحوه حذف اعداد

در مرحله بعدی متن‌ها را به توکن تبدیل می‌کنیم. برای انجام این کار از کتابخانه hazm استفاده می‌کنیم. ابتدا از کلاس WordTokenizer() یک شی به نام tokenizer درست می‌کنیم. سپس در متد apply() یک تابع بی‌نام ایجاد می‌کنیم که متن‌ها را به عنوان آرگومان دریافت می‌کند و به متد tokenize() از شی tokenizer پاس می‌دهد. متن توکن شده در ستونی به نام Tokenized_Content ذخیره می‌شود. شکل (۵۳) نحوه انجام این کار را نمایش می‌دهد.

```
Tokenizing the text

tokenizer = WordTokenizer()
df['Tokenized_Content'] = df['Content'].apply(lambda text: tokenizer.tokenize(text))
```

شکل ۵۳: توکنایز کردن متن

قدم بعدی حذف stop wordsهاست. برای حذف این کلمات، از فایل تکست به نام PersianStopWords ، موجود در فولدرهای دیتاست استفاده می‌کنیم. به این صورت که در یک تابع این فایل را می‌خوانیم، و کلمات موجود در آن را در یک مجموعه قرار می‌دهیم و سپس با استفاده از list comprehension صرفاً کلماتی را برمی‌گردانیم که در مجموعه stop words نیستند. شکل (۵۴) نحوه حذف stop wordsها را نمایش می‌دهد.

```
Removing stop words

# Read stop words from the text file
with open("PersianStopWords.txt", "r", encoding="utf-8") as file:
    stop_words = set(file.read().splitlines())

# Define a function to remove stop words
def remove_stop_words(tokens):
    return [word for word in tokens if word not in stop_words]

# Apply the function to the "Tokenized_Content" column
df['Tokenized_Content'] = df['Tokenized_Content'].apply(remove_stop_words)
```

شکل ۵۴: حذف stop wordsها

حال بررسی می‌کنیم که پنج توکنی که بیشتر از همه استفاده می‌شوند چه توکن‌هایی هستند. برای انجام این کار، ابتدا همه توکن ها را در یک لیست به نام all_tokens ذخیره می‌کنیم. سپس با استفاده از تابع Counter() تعداد هر توکن را شمارش می‌کنیم. سپس پنج توکن اول که بیشترین استفاده را داشته‌اند را نمایش می‌دهیم. شکل (۵۵) نحوه انجام این کار را نشان می‌دهد.

```
Five most frequently used tokens

# Flatten the list of tokens
all_tokens = [word for tokens in df['Tokenized_Content'] for word in tokens]

# Count the frequency of each word
word_counts = Counter(all_tokens)

# Get the five most common words
most_common_words = word_counts.most_common(5)

most_common_words

[('229758', 'کشور'),
 ('205930', 'سال'),
 ('196647', 'ایران'),
 ('131980', 'تهران'),
 ('116233', 'اسلامی')]
```

شکل ۵۵: نمایش ۵ توکنی که بیشتری استفاده را داشته‌اند

مشاهده می‌شود که پنج توکن کشور، سال، ایران، تهران، اسلامی به ترتیب بیشترین استفاده را داشته‌اند.

در قدم بعدی همه توکن‌های هر متن را به یک رشته تبدیل و سپس متن را نرمال می‌کنیم. نحوه اجرای نرمال سازی، دقیقاً شبیه نحوه توکن کردن متن است.

```
Join all tokens into one string

df['Tokenized_Content'] = df['Tokenized_Content'].apply(lambda tokens: ' '.join(tokens))

Normalizing the preprocessed text

normalizer = Normalizer()
df['Tokenized_Content'] = df['Tokenized_Content'].apply(lambda text: normalizer.normalize(text))
```

شکل ۵۶: تبدیل توکن‌ها به متن و نرمال سازی متن

در ادامه محتوای ستون Content را با محتوای Tokenized_Content جایگزین می‌کنیم. همچنین سطرهایی که مقادیر خالی دارند را حذف می‌کنیم.

TF-IDF-۳-۳

برای اجرای این روش از کلاس TfidfVectorizer() یک شی می‌سازیم. سپس با استفاده از متد fit_transform() و پاس دادن ستون Content به عنوان آرگومان، ماتریس هزار کلمه‌ای که بیشترین فرکانس را داشته‌اند را محاسبه می‌کنیم. سپس این ماتریس را به یک دیتافریم تبدیل می‌کنیم. شکل (۵۷) نحوه اجرای این کار را نمایش می‌دهد.

```
TF-IDF

# Create a TfidfVectorizer object
tfidf_vectorizer = TfidfVectorizer(max_features=1000)

# Fit and transform the 'Content' column of the DataFrame
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Content'])

# Convert to DataFrame (optional)
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
```

شکل ۵۷: اجرای TF-IDF

در مرحله بعدی، پنج کلمه‌ی مهم هر متن را نمایش می‌دهیم. شکل (۵۸) نحوه انجام این کار را نمایش می‌دهد.

همچنین پنج کلمه‌ای که در کل متن‌ها بیشترین مجموع مقدار را داشته‌اند را نمایش می‌دهیم. شکل (۵۹) نحوه انجام این کار را نشان می‌دهد.

```
top_terms_per_document = {}
for i, row in tfidf_df.iterrows():
    top_terms = row.sort_values(ascending=False).head(5).index.tolist()
    top_terms_per_document[i] = top_terms

# Print the top terms for each document
for document, top_terms in top_terms_per_document.items():
    print(f"Document {document}: {', '.join(top_terms)}")
```

```
Document 0: نقاشی، زندگی، آثار، موضوع، گروهی
Document 1: نمایشگاه، آثار، نمایش، خانه، هنری
Document 2: نقاشی، نمایشگاه، ساعت، تلفن، شماره
Document 3: بازی، بچه، خانه، کودکان، زندگی
Document 4: آموزشی، مدارس، آموزش، آموزان، دانش
Document 5: آذربایجان، جمهوری، مجلس، ملی، خواند
Document 6: دانش، آموزان، معرفی، نهایی، صرف
Document 7: دانشگاه، دانشجویان، علمی، اعتراض، صدور
Document 8: سبز، نیست، نیستند، هست، سوال
Document 9: دانش، آموزان، مختلف، نقاط، آموزش
Document 10: رشد، اقتصادی، سال، درصد، آینده
```

شکل ۵۸: نمایش ۵ کلمه مهم هر متن

Top five important terms used in the whole dataset

```
# Sum the TF-IDF values for each term across all documents
total_tfidf = tfidf_df.sum()

# Sort the terms based on their total TF-IDF values
top_terms = total_tfidf.sort_values(ascending=False).head(5)

# Print the top terms
print("Top five terms used in all documents:")
for term, tfidf in top_terms.items():
    print(f"{term}: Total TF-IDF = {tfidf}")
```

```
Top five terms used in all documents:
ایران: Total TF-IDF = 6274.548198367028
کشور: Total TF-IDF = 5875.753677587329
تهران: Total TF-IDF = 5288.305372489105
سال: Total TF-IDF = 5077.228962087621
تیم: Total TF-IDF = 4614.837217279096
```

شکل ۵۹: نمایش ۵ کلمه مهم در همه متون

حال به مصورسازی داده‌ها می‌پردازیم.

۳-۴-مصورسازی داده‌ها

برای انجام مصورسازی داده‌ها از گوگل کولب کمک می‌گیریم. اولین موردی که گوگل کولب در آن مفید واقع می‌شود این است که کتابخانه wordcould-fa که برای زبان فارسی ابر کلمه درست می‌کند، برای سیستم‌های لینوکسی توسعه داده شده است. دومین مورد این است که می‌توان در فضای گوگل کولب از کتابخانه polyglot استفاده کرد. کاربرد کتابخانه polyglot در اینجا، تحلیل عواطف متن است.

کدهای مربوط به این بخش در فایل Part3-2.ipnb قرار دارد

پس از آپلود فایل‌های اکسل روی گوگل درایو و فراخوانی و تبدیل آنها به یک دیتافریم، دیتافریم را طبق روش گفته شده پیش‌پردازش می‌کنیم. حال این دیتافریم برای مصورسازی آماده است.

از آنجایی که تولید ابر کلمات با همه مقالات از توان محاسباتی رایگان گوگل کولب خارج است، به صورت تصادفی پنجاه‌هزار متن را انتخاب می‌کنیم و از آنها ابر کلمات تولید می‌کنیم. تعداد پنجاه‌هزار با استفاده از آزمون و خطا به دست آمده است.

از کلاس WordCloudFa() یک شی به نام wc می‌سازیم. آرگومان‌های استفاده شده در ساخت این شی، طول و عرض عکس و آرگومانی به اسم no_reshape است. این آرگومان اطمینان حاصل می‌کند که کلمات به صورت به هم ریخته تولید نشوند.

در قدم بعدی تمامی متون استفاده شده را کنار هم قرار می‌دهیم و در متغیری به نام text ذخیره می‌کنیم. این متغیر را به متد generate() از شی text پاس می‌دهیم و نتیجه را در متغیر word_cloud ذخیره می‌کنیم. در نهایت با استفاده از متد to_image() عکس را در متغیر image قرار داده و با استفاده از متد save() عکس را به صورت png با نام persian-example.png ذخیره می‌کنیم. شکل (۶۰) نحوه انجام این فرایند و شکل (۶۱) خروجی نهایی ابر را نمایش می‌دهد.

```
random_sample = df.sample(n=50000)

wc = WordCloudFa(no_reshape=True, width=1200, height=800)

text = ' '.join(random_sample['Content'].astype(str))

word_cloud = wc.generate(text)

image = word_cloud.to_image()

image.save('persian-example.png')
```

شکل ۶۰: فرایند ایجاد wordcloud

حال برای بررسی عمیق‌تر داده‌ها و انجام مصورسازی‌های بیشتر از کتابخانه polyglot کمک می‌گیریم.

ابتدا پیش‌نیازهای این کتابخانه و خود آن را با استفاده از pip نصب می‌کنیم و سپس با دستور polyglot download LANG:fa! بسته زبان فارسی این کتابخانه را دریافت می‌کنیم. همچنین برای انجام تحلیل عواطف، بسته تحلیل عواطف فارسی را دریافت می‌کنیم.

شکل ۶۱: ابر کلمه تولید شده

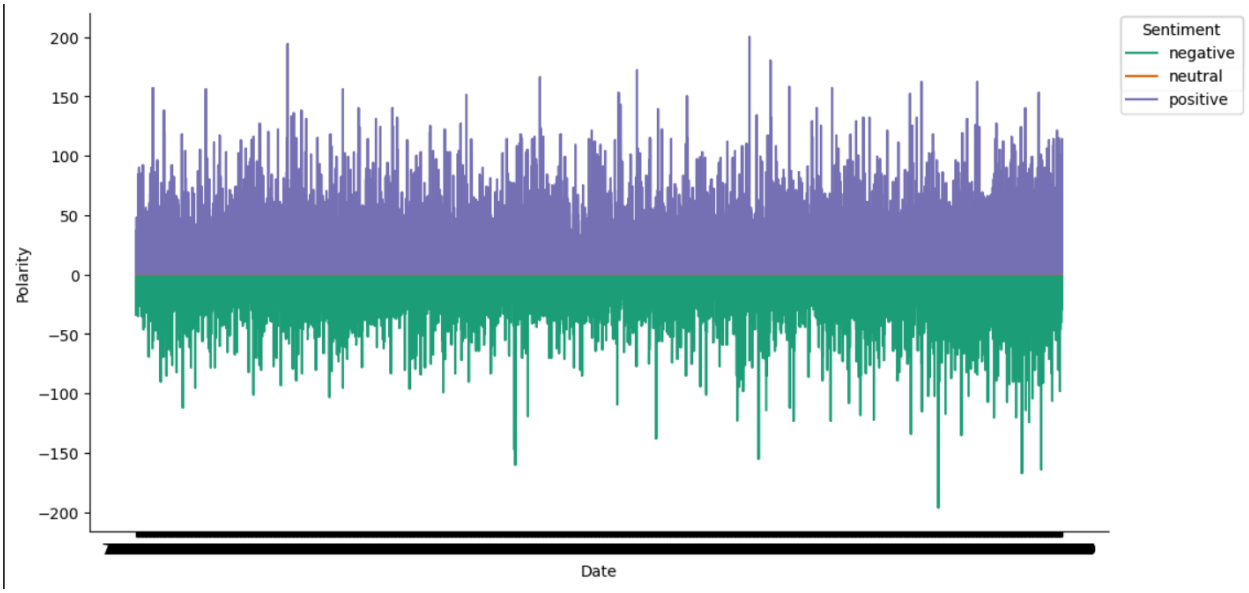
دستور دریافت این بسته polyglot download sentiment2.fa! است. نحوه کارکرد کلاس تحلیل عواطف این کتابخانه به این صورت است که جمله‌ها را بر اساس قطبیت آنها امتیاز بندی می‌کند. به طوری که جملاتی که بار مثبت دارند امتیاز قطبیت آنها بیشتر از صفر، جملاتی که بار منفی دارند، قطبیت آنها کمتر از صفر و جملات خنثی قطبیت صفر دارند.

رویکردی که برای تحلیل عواطف پیش گرفته می‌شود به این صورت است: ابتدا با استفاده از کتابخانه hazm، متون به جملات توکن‌بندی می‌شوند. سپس این لیست از توکن‌ها به صورت سطر به سطر به polyglot داده می‌شوند و این کتابخانه امتیاز قطبیت را جمله به جمله محاسبه و در لیستی ذخیره می‌کند سپس از این مقادیر میانگین گرفته می‌شود و به عنوان قطبیت نهایی متن اعلام می‌شود در صورت بروز خطا، مقدار None بازگردانده می‌شود. شکل (۶۲) نحوه انجام این کار توسط تابع analyze_sentiment() که لیست جملات را به عنوان آرگومان دریافت می‌کند نشان می‌دهد.

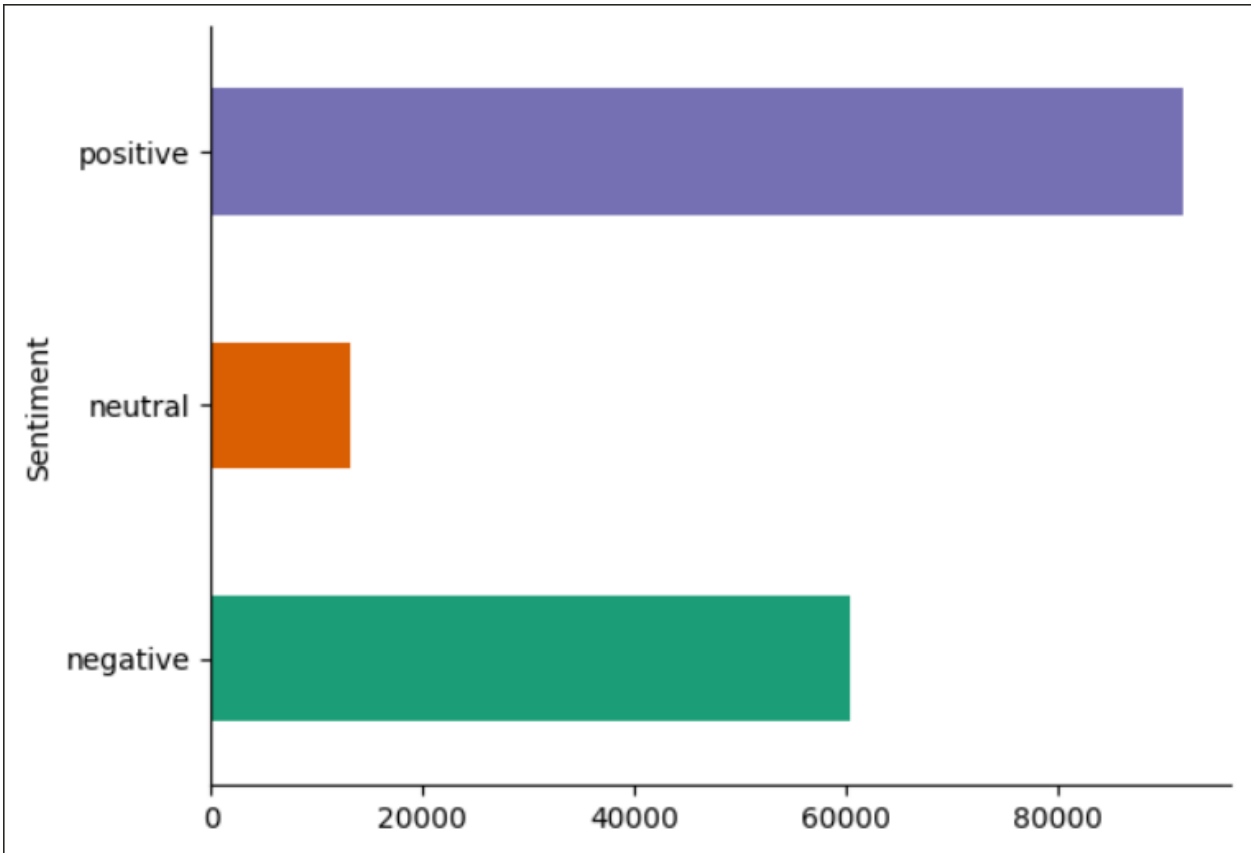
همچنین برای نشان دادن یک شمای کلی از وضعیت متن‌ها، قطبیت متن‌ها را گسسته سازی می‌کنیم، به طوری که اگر قطبیت متن‌ها مثبت باشد، در دسته positive، اگر منفی باشد در دسته negative و اگر خنثی باشد در دسته neutral قرار بگیرد. سپس به رسم نمودارها می‌پردازیم. شکل (۶۳) تا (۶۸) نمودارهای رسم شده از این دو ستون را نمایش می‌دهند.

```python
from polyglot.text import Text

def analyze_sentiment(sentences):
    try:
        avg_polarity = []

        # Iterate through each sentence in the list of sentences
        for text in sentences:
            # Create a Polyglot Text object from the input text with language set to Persian
            text_obj = Text(text)

            # Initialize variables to track sentiment polarity
            polarity = 0

            # Iterate through words in each sentence
            for sentence in text_obj.sentences:
                for word in sentence.words:
                    # Update polarity based on each word's polarity in the sentence
                    polarity += word.polarity

            # Append the polarity of the current sentence to the list
            avg_polarity.append(polarity)

        # Calculate the average polarity for all sentences in the row
        if avg_polarity:
            average_polarity = sum(avg_polarity) / len(avg_polarity)
        else:
            average_polarity = None

        return average_polarity
    except Exception as e:
        # Handle any errors that may occur during sentiment analysis
        print(f"Error occurred: {e}")
        return None
```

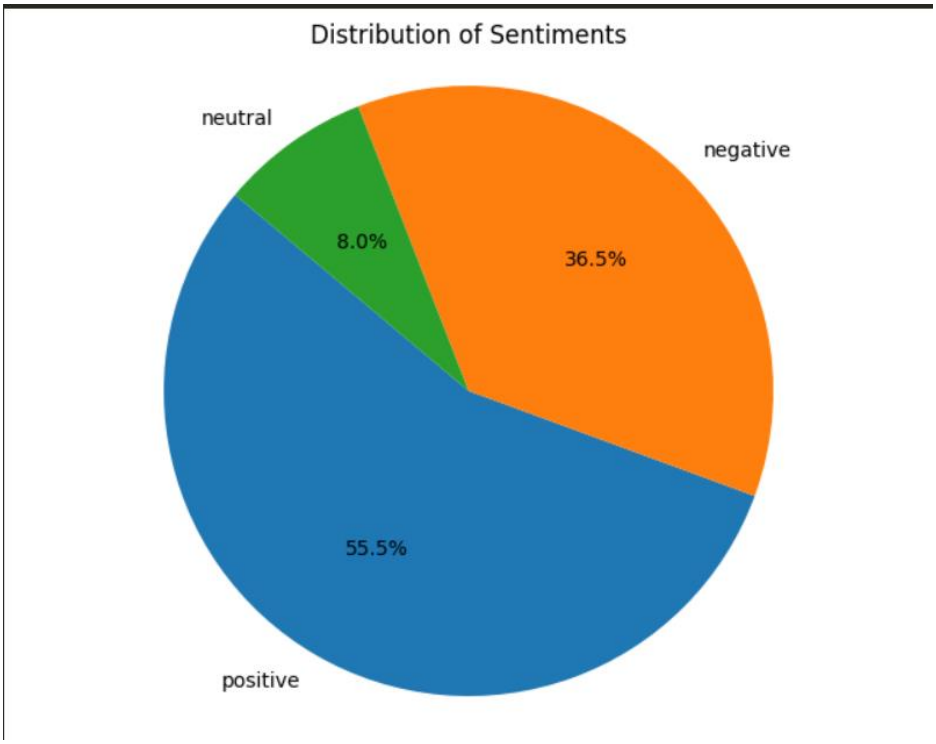
شکل ۶۲: انجام تحلیل عواطف با استفاده از کتابخانه polyglot



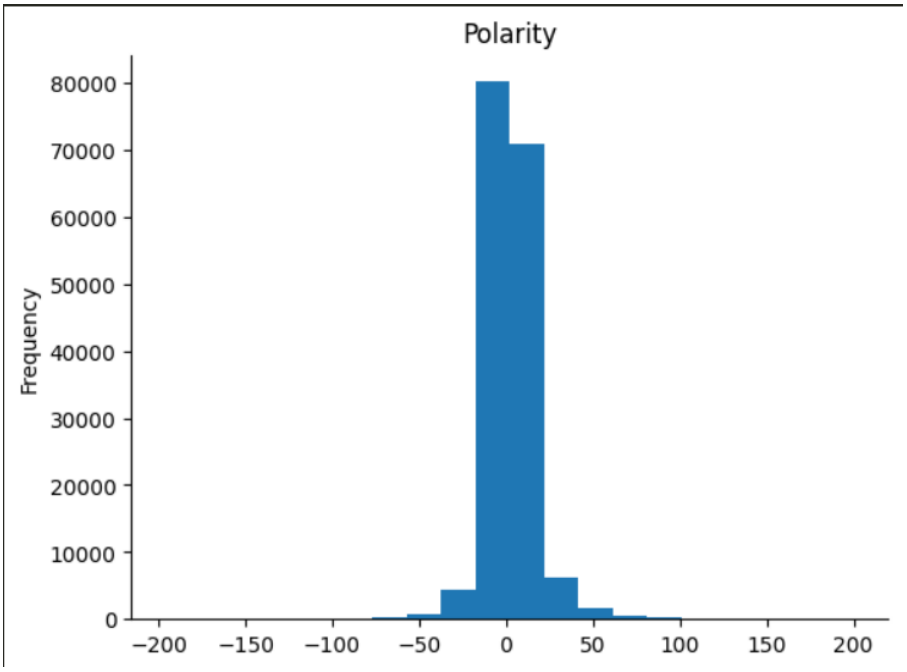
شکل ۶۳: نمودار سری زمانی



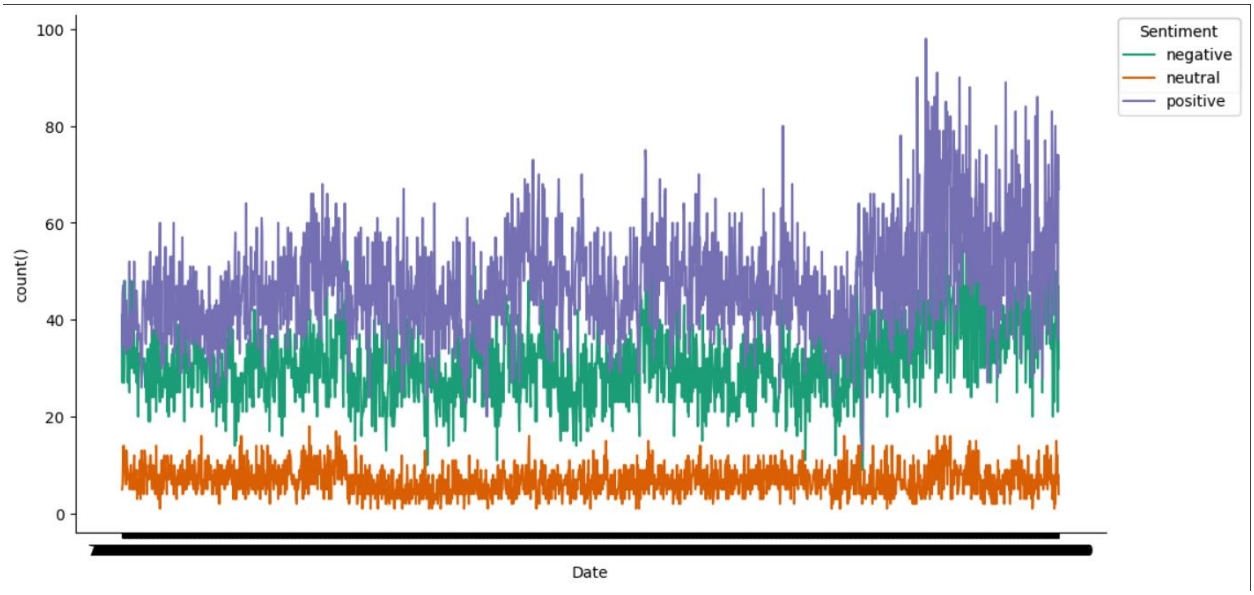
شکل ۶۴: نمودار میله‌ای تعداد متن‌های مثبت، خنثی و منفی



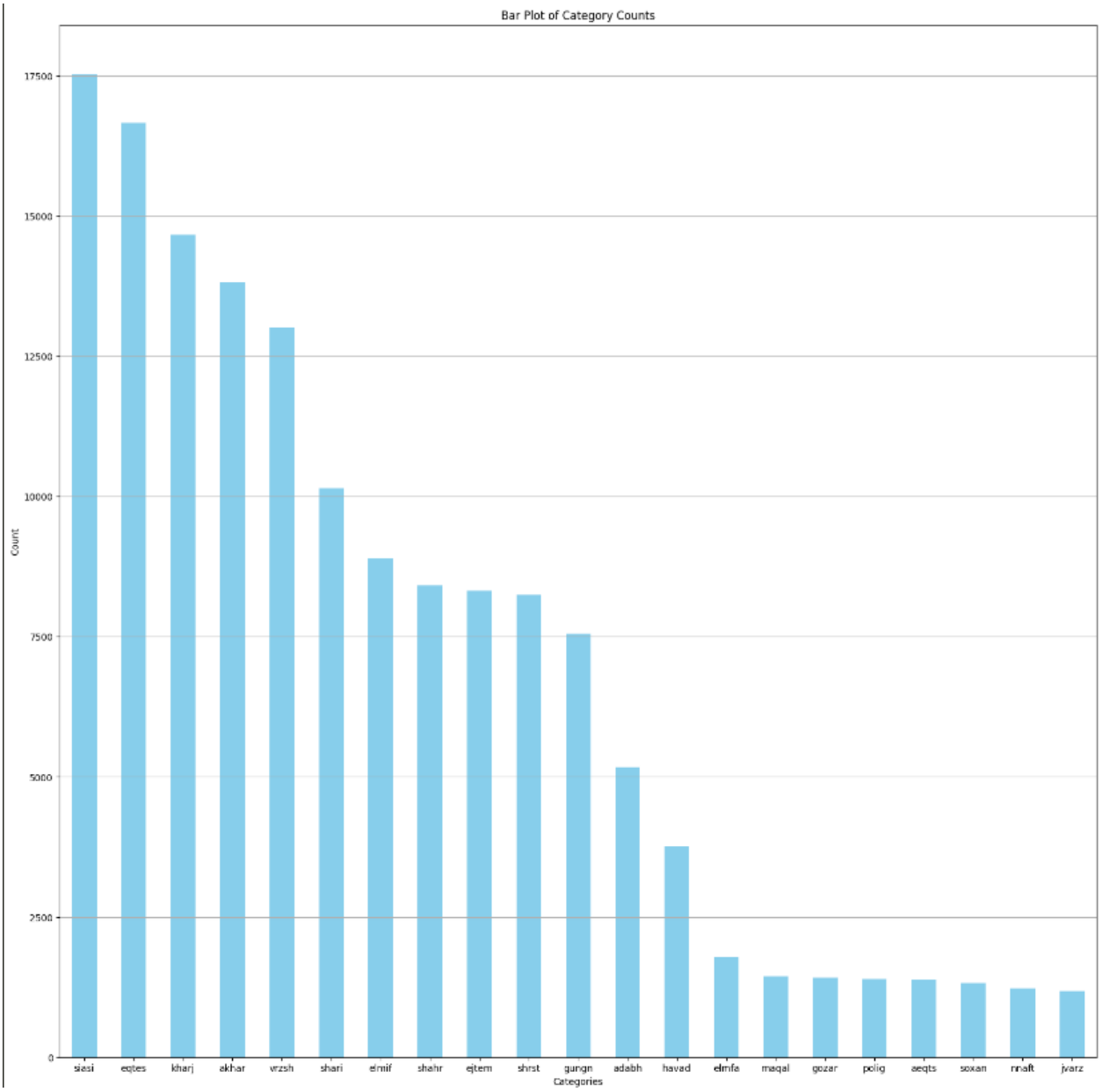
شکل ۶۵: نمودار سهم هر دسته از قطبیت‌ها از کل متون



شکل ۶۶: هسیتوگرام میزان قطبیت



شکل ۶۷: نمودار تعداد اخبار مثبت، خنثی و منفی در طول زمان



شکل ۶۸: نمودار میله‌ای تعداد متون در بیست دسته اول موضوعات

۴-پیوست

همچنین این لینک در یک فایل تکست در کنار سایرفایل‌ها قرار داده شده است.

https://drive.google.com/drive/folders/1JsOvNBHwrfdyz0zZ9XGDOM8B6wj6ZTI-?usp=sharing