

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تکلیف دوم درس یادگیری ماشین کاربردی

استاد درس: دکتر ناظرفرد

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: ۴۰۲۱۲۵۰۰۵

اللهُ أَكْبَرُ
لِلّهِ الْحَمْدُ
لِلّهِ الْحَمْدُ
لِلّهِ الْحَمْدُ

أ

فهرست مطالب

۱	۱	-پاسخ سوال اول
۱	۱	-۱-لود کردن دیتابست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی دادهها
۴	۱	-۲-تصویر سازی دادهها
۴	۱	-۱-۱-رسم و تفسیر هیستوگرام
۶	۱	-۲-۱-رسم نمودار دایره‌ای و تفسیر آن
۷	۱	-۲-۲-رسم و تفسیر نمودارهای جعبه‌ای
۹	۱	-۲-۳-رسم و تفسیر ماتریس همبستگی
۱۰	۱	-۲-۴-رسم و تفسیر نمودار همبستگی با ستون هدف
۱۲	۱	-۳-تشخیص داده‌های پرت
۱۲	۱	-۳-۱-جایگزینی داده‌ها با حد پایین یا حد بالا
۱۵	۱	-۳-۲-حذف داده‌های پرت
۱۵	۱	-۴-پر کردن مقادیر خالی
۱۵	۱	-۴-۱-تعريف شی پیش‌پردازش برای پر کردن مقادیر خالی با KNN
۱۶	۱	-۴-۲-تعريف شی پیش‌پردازش برای پر کردن مقادیر خالی با میانه
۱۷	۱	-۴-۳-انتخاب ویژگی
۱۷	۱	-۴-۱-انتخاب ویژگی با استفاده از روش PCA
۱۸	۱	-۴-۲-انتخاب ویژگی با روش Mutual Information
۲۳	۱	-۴-۳-پاسخ سوال دوم
۲۴	۱	-۴-۱-پر کردن مقادیر خالی
۲۷	۱	-۴-۲-استفاده از SelectKBest
۲۷	۱	-۴-۳-آموزش KNN
۲۹	۱	-۴-۴-آموزش و تست مدل با فاصله‌های متفاوت
۲۹	۱	-۴-۱-فاصله منهن
۳۰	۱	-۴-۲-فاصله کسینوسی
۳۰	۱	-۴-۳-انجام پیش‌پردازش بیشتر برای رسیدن به نتایج بهتر
۳۲	۱	-۴-۴-پاسخ سوال سوم
۳۲	۱	-۴-۱-پیش‌پردازش دادهها

۳۲.....	۱-۱-۳- حذف کاراکترهای n
۳۲.....	۱-۲- حذف کاراکترهای انگلیسی
۳۲.....	۱-۳- حذف علامه نگارشی
۳۴.....	۱-۴- نرمال کردن متنها
۳۴.....	۱-۵- توکن کردن متن
۳۴.....	۱-۶- حذف ایستوازه‌ها (کلمات پالایشی یا stop words)
۳۵.....	۱-۷- آموزش KNN و پیدا کردن بهترین K
۳۸.....	۱-۸- پاسخ سوال چهارم
۳۸.....	۱-۹- وارد کردن و پیش‌پردازش داده‌ها
۴۱.....	۱-۱۰- آموزش KNN
۴۱.....	۱-۱۱- استفاده از Cross Validation

فهرست شکل‌ها

شکل ۱: وارد کردن کتابخانه‌های مورد نیاز	۱
شکل ۲: خواندن دیتاست	۱
شکل ۳: کد مربوط به نمایش ده سطر تصادفی	۲
شکل ۴: ده سطر تصادفی نمایش داده شده	۲
شکل ۵: کد مربوط به نشان دادن تعداد مقادیر گمشده	۲
شکل ۶: تعداد مقادیر خالی در هر سطر	۲
شکل ۷: کد بررسی تعداد سطرهای تکراری	۳
شکل ۸: کد بررسی تعداد صفرها در هر ستون	۳
شکل ۹: بررسی تعداد و جمعیت موجود در سطرهای گستته	۳
شکل ۱۰: خروجی کد شکل ۹	۹
شکل ۱۱: انتقال سطر هدف به سطر آخر	۴
شکل ۱۲: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا	۴
شکل ۱۳: کد مربوط به رسم هیستوگرام	۵
شکل ۱۴: هیستوگرام‌های رسم شده	۵
شکل ۱۵: کد رسم نمودار دایره‌ای	۶
شکل ۱۶: نمودار دایره‌ای	۷
شکل ۱۷: کد رسم کننده نمودارهای جعبه‌ای	۸
شکل ۱۸: نمودارهای جعبه‌ای	۸
شکل ۱۹: آماده‌سازی دیتا فریم برای محاسبه ماتریس همبستگی	۹

ت

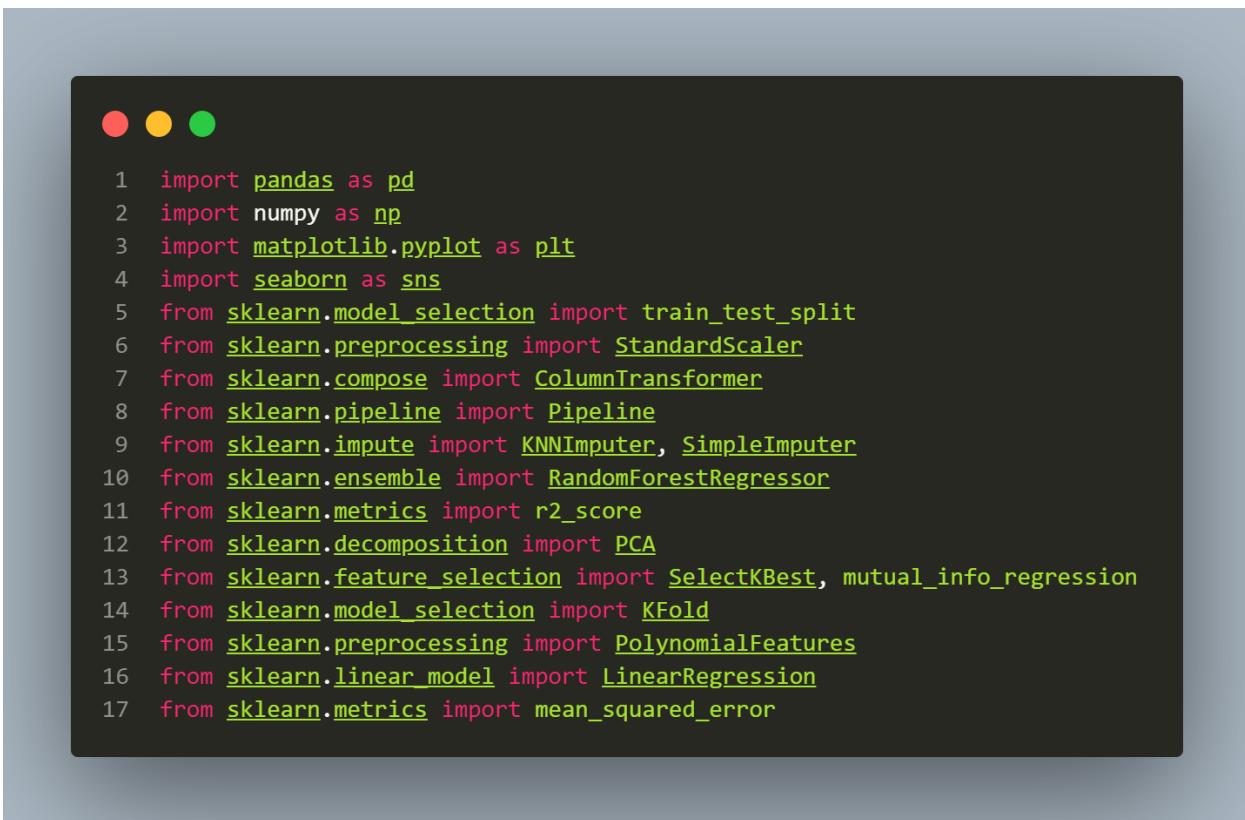
۹ شکل ۲۰: رسم ماتریس همبستگی
۱۰ شکل ۲۱: ماتریس همبستگی رسم شده
۱۱ شکل ۲۲: کد رسم کننده نمودار میزان همبستگی با سطر هدف
۱۱ شکل ۲۳: نمودار میزان همبستگی با سطر هدف
۱۳ شکل ۲۴: جایگزینی داده‌های پرت با حد بالا و حد پایین
۱۳ شکل ۲۵: ایجاد شی از کلاس ColumnTransformer
۱۳ شکل ۲۶: پیش‌پردازش داده‌های آموزشی
۱۴ شکل ۲۷: پیش‌پردازش داده‌های تست
۱۴ شکل ۲۸: تعریف مدل
۱۴ شکل ۲۹: آموزش و تست مدل
۱۵ شکل ۳۰: محاسبه و نمایش $2R$
۱۵ شکل ۳۱: حذف داده‌های پرت
۱۶ شکل ۳۲: ایجاد شی از کلاس پایپ‌لاین
۱۶ شکل ۳۳: ایجاد شی از کلاس ColumnTransformer
۱۷ شکل ۳۴: پایپ‌لاین به روز شده
۱۷ شکل ۳۵: پایپ‌لاین ایجاد شده برای PCA
۱۸ شکل ۳۶: اجرای پایپ‌لاین روی داده‌های آموزشی
۱۸ شکل ۳۷: انتخاب نه ویژگی برتر با استفاده از Mutual Information
۱۹ شکل ۳۸: رسم نمودار از میانگین MSE هر درجه
۲۰ شکل ۳۹: ارزیابی درجه‌های رگرسیون با استفاده از CV
۲۰ شکل ۴۰: نمودار MSE به ازای درجه
۲۱ شکل ۴۱: پایپ‌لاین پیش‌پردازش و آموزش رگرسیون
۲۱ شکل ۴۲: ایجاد ویژگی جدید با دو ویژگی قدیمی
۲۱ شکل ۴۳: نمودار MSE بر درجه برای حالت جدید دیتاست
۲۲ شکل ۴۴: نمودار MSE براساس درجه در حالت داشتن ویژگی Rooms_per_household
۲۳ شکل ۴۵: خروجی متدهای unique()
۲۳ شکل ۴۶: استفاده از method chaining
۲۴ شکل ۴۷: تعداد مقادیر خالی در هر ستون
۲۴ شکل ۴۸: ذخیره نام ستون‌ها در دو لیست مجزا
۲۴ شکل ۴۹: ایجاد پایپ‌لاین برای داده‌های عددی
۲۵ شکل ۵۰: پایپ‌لاین برای مقادیر دسته‌ای
۲۵ شکل ۵۱: شی ساخته شده از ColumnTransformer
۲۵ شکل ۵۲: پایپ‌لاین نهایی
۲۶ شکل ۵۳: انجام Cross Validation و محاسبه $2R$
۲۶ شکل ۵۴: پایپ‌لاین به روز شده برای مقادیر عددی
۲۷ شکل ۵۵: انتخاب ویژگی با استفاده از SelectKBest

۲۸.....	شکل ۵۶: کردن داده‌های آموزشی Oversample
۲۸.....	شکل ۵۷: ماتریس درهم ریختگی مدل آموزش داده شده
۲۹.....	شکل ۵۸: پایپ‌لاین آموزش/تست با فاصله متهمن
۲۹.....	شکل ۵۹: ماتریس درهم ریختگی KNN با فاصله منهمن
۳۰.....	شکل ۶۰: پایپ‌لاین تست و آموزش KNN با فاصله کسینوسی
۳۰.....	شکل ۶۱: ماتریس درهم ریختگی مدل KNN با فاصله کسینوسی
۳۱.....	شکل ۶۲: حذف داده‌های پرت
۳۲.....	شکل ۶۳: تعداد ایمیل‌های با لیبل سالم و اسپم
۳۲.....	شکل ۶۴: حذف کاراکتر \n
۳۳.....	شکل ۶۵: حذف کاراکترهای انگلیسی
۳۳.....	شکل ۶۶: حذف علام نگارشی
۳۳.....	شکل ۶۷: حذف اعداد با استفاده از رجکس
۳۴.....	شکل ۶۸: نحوه انجام نرمال سازی متن
۳۴.....	شکل ۶۹: توکن سازی متن
۳۵.....	شکل ۷۰: پیدا کردن و حذف ایستوازه‌ها
۳۵.....	شکل ۷۱: بازتولید متن‌ها از کلمه‌ها
۳۶.....	شکل ۷۲: استفاده از روش TF-IDF
۳۶.....	شکل ۷۳: تعریف مدل و دیکشنری پارامترها
۳۷.....	شکل ۷۴: کد رسم نمودار دقت بر اساس K
۳۷.....	شکل ۷۵: نمودار دقت بر اساس K
۳۸.....	شکل ۷۶: وارد کردن کتابخانه‌های مورد نیاز
۳۸.....	شکل ۷۷: تعریف الگوی دریافتی
۳۹.....	شکل ۷۸: خواندن و انتخاب تصادفی فایل‌ها
۳۹.....	شکل ۷۹: پیش‌پردازش یک به یک عکس‌ها
۴۰.....	شکل ۸۰: تابع خواندن و پیش‌پردازش عکس‌ها
۴۰.....	شکل ۸۱: فراخوانی و پیش‌پردازش عکس‌ها و ایجاد دیتاست نهایی
۴۱.....	شکل ۸۲: کد نمایش گزارش عملکرد
۴۱.....	شکل ۸۳: گزارش عملکرد مدل KNN
۴۲.....	شکل ۸۴: دیکشنری محاسبه معیارهای ارزیابی
۴۲.....	شکل ۸۵: انجام Cross Validation
۴۲.....	شکل ۸۶: نتایج به دست آمده از Cross Validation

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 ایمپورت 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. به مظور بازنولید نتایج مشابه تعریف می‌شود. شکل (۳)، خواندن دیتاست و شکل (۳) نشان دادن ده ردیف رندوم را نمایش می‌دهد.



```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.model_selection import train_test_split
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.compose import ColumnTransformer
8 from sklearn.pipeline import Pipeline
9 from sklearn.impute import KNNImputer, SimpleImputer
10 from sklearn.ensemble import RandomForestRegressor
11 from sklearn.metrics import r2_score
12 from sklearn.decomposition import PCA
13 from sklearn.feature_selection import SelectKBest, mutual_info_regression
14 from sklearn.model_selection import KFold
15 from sklearn.preprocessing import PolynomialFeatures
16 from sklearn.linear_model import LinearRegression
17 from sklearn.metrics import mean_squared_error
```

شکل ۱: وارد کردن کتابخانه‌های مورد نیاز



```
1 df = pd.read_csv("housing.csv")
```

شکل ۲: خواندن دیتاست

```

1 # Set random seed for reproducibility
2 np.random.seed(42)
3 random_rows = df.sample(n=10)
4 random_rows

```

شکل ۳: کد مربوط به نمایش ده سطر تصادفی

همچنین شکل (۴) ده سطر تصادفی نمایش داده شده را نشان می‌دهد.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN
13311	-117.61	34.08	12.0	4427.0	NaN	2400.0	843.0	4.7147	158700.0	INLAND
7113	-118.02	33.89	36.0	1375.0	NaN	670.0	221.0	5.0839	198200.0	<1H OCEAN
7668	-118.08	33.92	38.0	1335.0	NaN	1011.0	269.0	3.6908	157500.0	<1H OCEAN
18246	-122.08	37.39	4.0	2292.0	NaN	1050.0	584.0	4.8036	340000.0	NEAR BAY
5723	-118.23	34.18	45.0	2332.0	NaN	943.0	339.0	8.1132	446600.0	<1H OCEAN

شکل ۴: ده سطر تصادفی نمایش داده شده

در ادامه با استفاده از متod info() نگاهی کلی به ستون‌های باقی‌مانده می‌اندازیم. سپس با استفاده از method chaining از نمایش داده شده در شکل (۵) بررسی می‌کنیم که در هر ستون در مجموع چه تعداد مقادیر تعریف نشده وجود دارد.

```

1 df.isnull().sum()

```

شکل ۵: کد مربوط به نشان دادن تعداد مقادیر گمشده

با توجه به شکل (۶) مشاهده می‌شود که ستون total_bedrooms دارای مقادیر خالی است که در ادامه به مدیریت آنها خواهیم پرداخت.

longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	207
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity	0
dtype:	int64

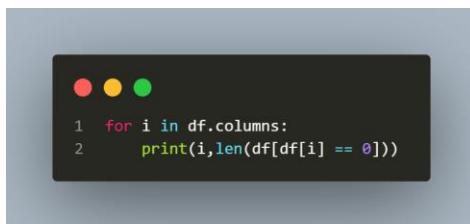
شکل ۶: تعداد مقادیر خالی در هر سطر

در ادامه سه مورد را بررسی می‌کنیم: اول اینکه چه تعداد سطر تکراری داریم. شکل (۷) نحوه انجام این کار را نمایش می‌دهد. سپس بررسی می‌کنیم که آیا در ستون‌هایی که مقدار صفر معنایی ندارد، رکوردی با مقدار صفر داریم یا خیر. شکل (۸) نحوه انجام این کار را نمایش می‌دهد.



```
1 df.duplicated().sum()
```

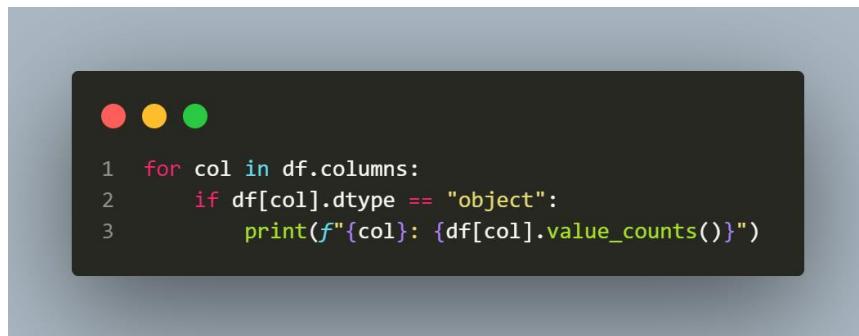
شکل ۷: کد بررسی تعداد سطرهای تکراری



```
1 for i in df.columns:
2     print(i,len(df[df[i] == 0]))
```

شکل ۸: کد بررسی تعداد صفرها در هر ستون

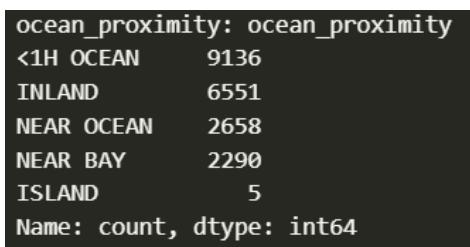
با بررسی خروجی این دو کد، متوجه می‌شویم که هیچ سطر تکراری در دیتافریم وجود ندارد. همچنین در هیچ یک از ستون‌ها مقدار صفر نداریم. در قدم بعدی بررسی می‌کنیم که در سطرهایی که نوع داده‌ی دسته‌ای دارد، چه کلاس‌هایی موجود داریم و جمعیت هر دسته چقدر است.



```
1 for col in df.columns:
2     if df[col].dtype == "object":
3         print(f'{col}: {df[col].value_counts()}'")
```

شکل ۹: بررسی تعداد و جمعیت موجود در سطرهای گستته

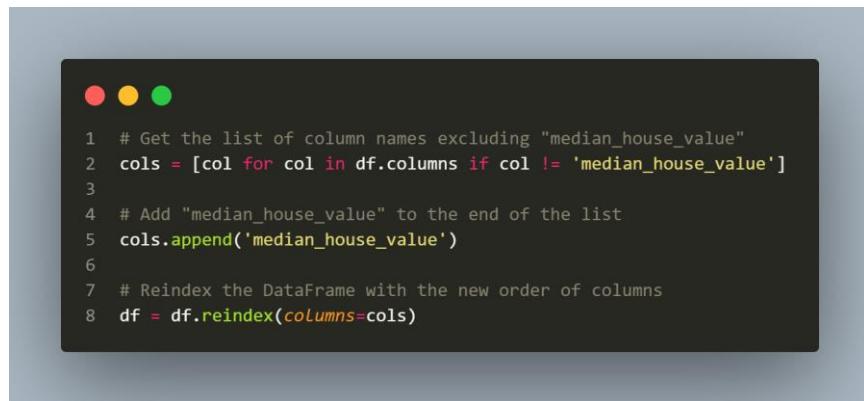
شکل (۱۰) خروجی این کد را نمایش می‌دهد. مشاهده می‌شود که دسته‌ای با نام غیر معمول در ستون ocean_proximity وجود ندارد.



```
ocean_proximity: ocean_proximity
<1H OCEAN      9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND          5
Name: count, dtype: int64
```

شکل ۱۰: خروجی کد شکل ۹

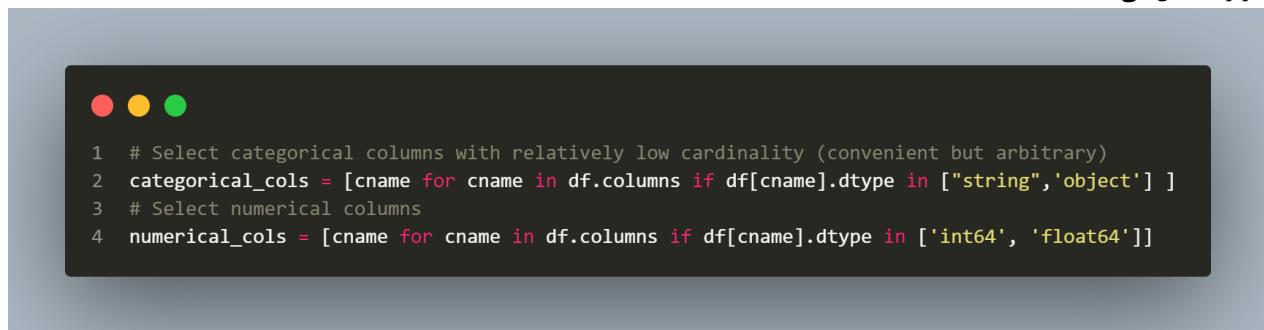
حال برای حفظ قرارداد، سطر هدف را به سطر آخر دیتا فریم انتقال می‌دهیم. شکل (۱۱) کدی که این کار را بر عهده دارد نمایش می‌دهد.



```
1 # Get the list of column names excluding "median_house_value"
2 cols = [col for col in df.columns if col != 'median_house_value']
3
4 # Add "median_house_value" to the end of the list
5 cols.append('median_house_value')
6
7 # Reindex the DataFrame with the new order of columns
8 df = df.reindex(columns=cols)
```

شکل ۱۱: انتقال سطر هدف به سطر آخر

حال یک تقسیم‌بندی انجام می‌دهیم که در ادامه‌ی کار مفید واقع می‌شود. با استفاده از List comprehension نام ستون‌هایی که داده‌های عددی دارند را در یک لیست و نام ستون‌هایی که دارای مقادیر دسته‌ای هستند را در یک لیست دیگر قرار می‌دهیم. شکل (۱۲) نحوه انجام این کار را نمایش می‌دهد.



```
1 # Select categorical columns with relatively low cardinality (convenient but arbitrary)
2 categorical_cols = [cname for cname in df.columns if df[cname].dtype in ["string", 'object']]
3 # Select numerical columns
4 numerical_cols = [cname for cname in df.columns if df[cname].dtype in ['int64', 'float64']]
```

شکل ۱۲: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا

۱-۲-۱-تصویر سازی داده‌ها

۱-۲-۱-رسم و تفسیر هیستوگرام

شکل ۱۳ نحوه رسم هیستوگرام را نمایش می‌دهد. num_cols و num_rows ابعاد شبکه فرعی را مشخص می‌کنند. در این مورد، یک شبکه سه در سه ایجاد می‌کند، بنابراین می‌توان تا ۹ نمودار فرعی را در خود جای داد.تابع plt.subplots() برای تولید یک شکل و مجموعه‌ای از نمودارهای فرعی (محور) استفاده می‌شود. (۰،۱۰) ۲۰ اندازه کل شکل figsize (۰،۱۰) اینچ عرض و ۱۰ اینچ ارتفاع) را مشخص می‌کند.

سپس حلقه روی هر ستون مشخص شده در لیست numerical_cols از df تکرار می‌شود. برای هر ستون: sns.histplot() یک هیستوگرام از ستون ایجاد می‌کند و آن را بر روی نمودار فرعی مربوطه ترسیم می‌کند که هر هیستوگرام باید ۲۰ میله داشته باشد. سپس عنوان هر نمودار بالای آن مشخص می‌شود.

سپس لوب بعدی هر فضای خالی که در آن نمودار وجود ندارد را حذف می‌کند. plt.tight_layout() نمودارهای فرعی و برجسب‌ها را طوری تنظیم می‌کند که به خوبی در ناحیه شکل قرار گیرند و روی هم قرار نگیرند. plt.show() شکل کامل را با تمام نمودارهای فرعی نمایش می‌دهد.

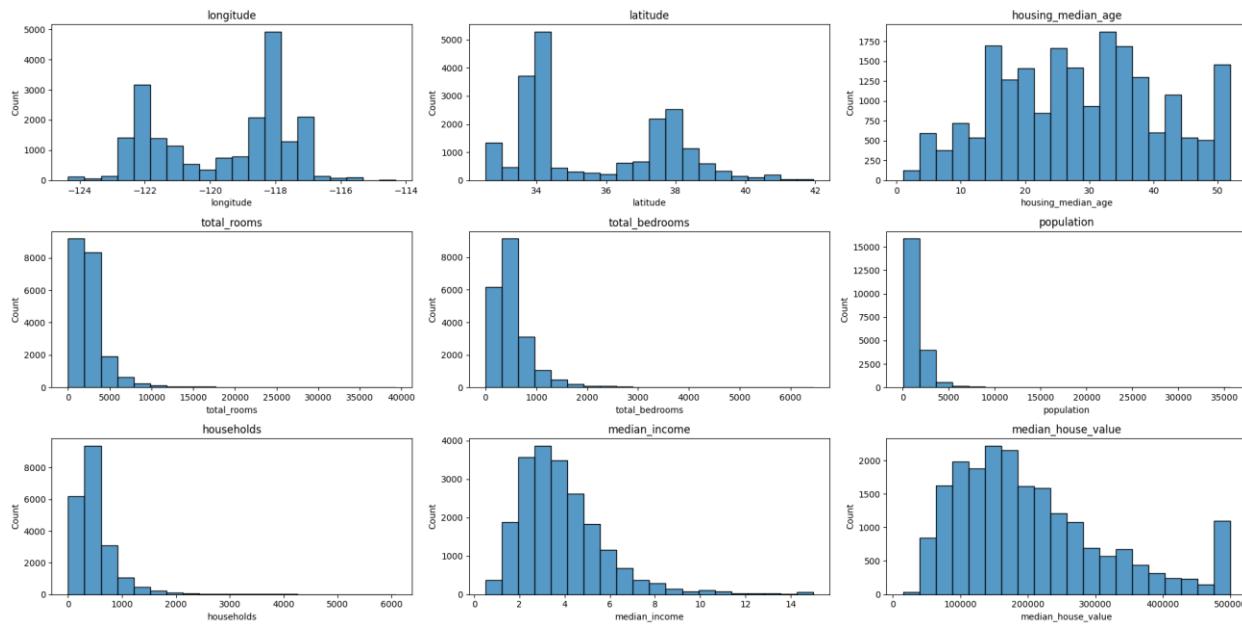
```

1 # Calculate the number of rows and columns needed for subplots
2 num_rows = 3
3 num_cols = 3
4
5 # Create subplots
6 fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 10))
7
8 # Flatten the axes array for easy iteration
9 axes = axes.flatten()
10
11 # Plot histograms for each column
12 for i, col in enumerate(df.loc[:, numerical_cols]):
13     sns.histplot(df[col], ax=axes[i], bins=20)
14     axes[i].set_title(col) # Set the title to the column name
15
16 # Hide any remaining empty subplots
17 for i in range(len(df.columns), len(axes)):
18     axes[i].axis('off')
19
20 # Adjust layout to prevent overlap
21 plt.tight_layout()
22
23 # Show the plot
24 plt.show()

```

شکل ۱۳: کد مربوط به رسم هیستوگرام

همچنین شکل (۱۴) هیستوگرام‌های رسم شده را نمایش می‌دهد.



شکل ۱۴: هیستوگرام‌های رسم شده

تفسیر:

:Longitude

توزیع چند وجهی است، که نشان دهنده چندین قله است که در آن داده های مسکن به صورت طولی متتمرکز شده اند. این نشان می دهد که خوشه های متعددی از خانه ها در طول های جغرافیایی مختلف، احتمالاً مربوط به شهرها یا مناطق مختلف است.

Latitude

مشابه طول جغرافیایی، توزیع چند وجهی با تمرکز داده های مسکن در عرض های جغرافیایی خاص است. این خوشة ها را در مناطق جغرافیایی خاصی نشان می دهد، که وقتی با طول جغرافیایی ترکیب می شوند، می توانند مناطق جغرافیایی خاصی را مشخص کنند.

Housing Median Age

این هیستوگرام توزیع نسبتاً یکنواختی با قله های متعدد نشان می دهد که نشان می دهد خانه هایی در طیف وسیعی از سنین وجود دارد. برای خانه های بسیار جدید و بسیار قدیمی، با تعداد خانه های بیشتر در محدوده سنی متوسط، افت جزئی وجود دارد.

Total Rooms

چوگنی به سمت راست است، به این معنی که بیشتر نقاط داده در سمت چپ قرار دارند، با تعداد خانه های کمتری که تعداد کل اتاق ها بسیار زیاد است.

تعداد کمی از خانه ها با تعداد بسیار زیاد اتاق وجود دارد که به صورت پرت به نظر می رسد.

Total Bedrooms

مشابه کل اتاق ها، این هیستوگرام نیز دارای انحراف راست است.

اکثر خانه ها اتاق خواب های کمتری دارند و با افزایش تعداد اتاق خواب ها، تعداد آنها کاهش می یابد.

Population

هیستوگرام جمعیت نیز دارای انحراف راست است، که مناطق زیادی با جمعیت کوچکتر و مناطق کمتر با جمعیت بسیار زیاد را نشان می دهد.

Households

این یکی دیگر از توزیع های منحرف به راست است که با افزایش تعداد خانوارها، کاهش شدیدی دارد. بیشتر نقاط داده در ناحیه ای متمرکز شده اند که خانوارهای کمتری را نشان می دهد.

Median Income

به نظر می رسد توزیع میانه درآمد کمی چوگن به راست است، اما به طور نرمال تر از سایر ویژگی ها توزیع می شود، که نشان می دهد سطوح درآمد به طور یکنواخت تر در میان مجموعه داده ها توزیع می شود، البته با دنباله ای به سمت درآمدهای بالاتر.

Median House Value

هیستوگرام مقدار متوسط خانه یک توزیع چند وجهی با چندین قله را نشان می دهد، که نشان می دهد خانه ها در دسته های ارزش مجزا قرار می گیرند. این نشان دهنده قیمت های متنوع مسکن، با اوج ها در محدوده های ارزشی خاص است.

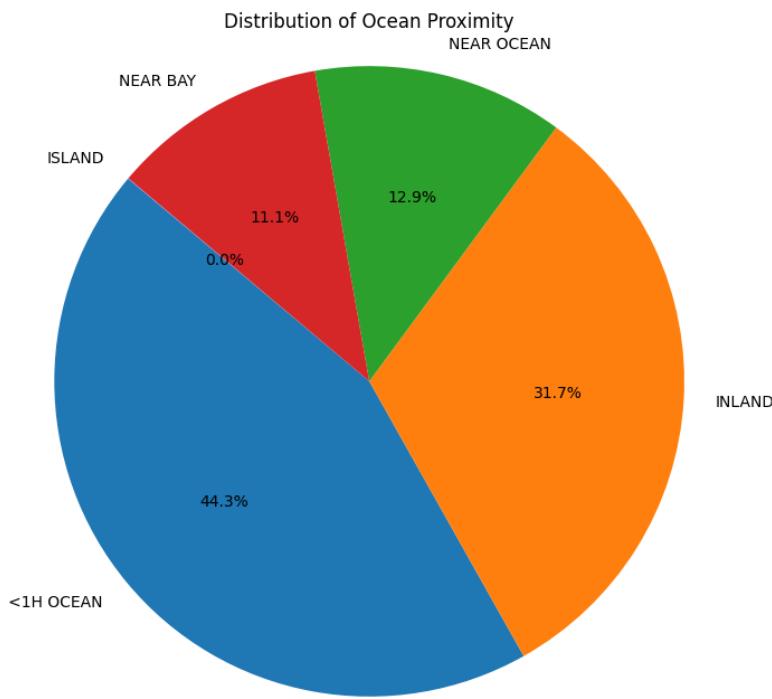
۲-۱- رسم نمودار دایره ای و تفسیر آن

برای تنها ستون داده های دسته ای این دیتافرید یک نمودار دایره ای رسم می کنیم. نحوه رسم این نمودار به این صورت است که ابتدا جمعیت هر دسته را با متد `value_counts()`. محاسبه می کنیم سپس با استفاده از کتابخانه `matplotlib` نمودار را رسم می کنیم. شکل (۱۵) نمودار رسم شده را نمایش می دهد.



شکل ۱۵: کد رسم نمودار دایره ای

همچنین شکل (۱۶) خروجی این کد را نمایش می‌دهد.



شکل ۱۶: نمودار دایره‌ای

تفسیر:

نزدیکی به اقیانوس: داده‌ها نشان می‌دهند که طبقه‌بندی واضحی از مسکن‌ها بر اساس نزدیکی به اقیانوس وجود دارد و اکثر خانه‌ها به اقیانوس نزدیک هستند.

همچنین بخش قابل توجهی از خانه‌ها از اقیانوس دور هستند که نشان می‌دهد دیتابست پراکندگی خوبی دارد.
خانه‌های ساخته شده در این مجموعه داده یا وجود ندارند یا بسیار نادر هستند.

۱-۲-۳-رسم و تفسیر نمودارهای جعبه‌ای

رسم نمودارهای جعبه‌ای برای ستونهای عددی همانند رسم هیستوگرام‌ها است با این تفاوت که ازتابع (boxplot) برای رسم نمودار استفاده می‌شود. شکل (۱۷) کد رسم کننده این نمودارها را نمایش می‌دهد. همچنین شکل (۱۸) خروجی این کد را نمایش می‌دهد.

تفسیر:

Longitude and Latitude

این دو نمودار موقعیت جغرافیایی املاک را نشان می‌دهند. جعبه‌ها طول کمی دارند و محدوده متغیرکزی از مقادیر را با چند نقطه پرت نشان می‌دهند. این گسترش حاکی از یک منطقه جغرافیایی محدود در حال مطالعه است، مانند یک منطقه یا ایالت خاص.

Housing Median Age

توزیع نسبتاً یکنواخت به نظر می‌رسد، با تمرکز جزئی در سنین میانی. این نشان دهنده تنوع سن خانه‌هاست.
: Households و Total Rooms, Total Bedrooms, Population,

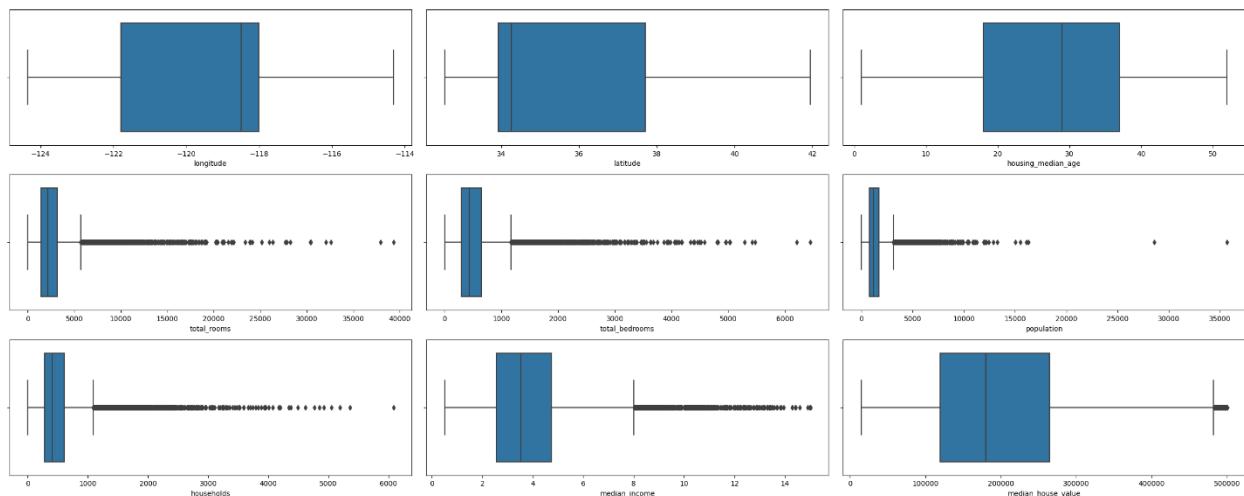
این نمودارها توزیعی به سمت راست را نشان می‌دهند که از سبیل‌های بلندی که به سمت راست امتداد می‌یابند و خط میانی به سمت چپ جعبه مشهود است.

```

1 import seaborn as sns
2 import matplotlib.pyplot as plt
3
4 # Calculate the number of rows and columns needed for subplots
5 num_rows = 3
6 num_cols = 3 # Number of numerical columns in your DataFrame
7
8 # Create subplots
9 fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 10))
10
11 # Plot boxplots for each column
12 for i, col in enumerate(numerical_cols):
13     sns.boxplot(x=df[col], ax=axes[i // num_cols, i % num_cols]) # Adjusted indexing
14
15 # Set labels for each subplot
16 for i, col in enumerate(numerical_cols):
17     axes[i // num_cols, i % num_cols].set_xlabel(col) # Adjusted indexing
18
19 # Hide any remaining empty subplots
20 for i in range(len(numerical_cols), num_rows * num_cols):
21     axes[i // num_cols, i % num_cols].axis('off')
22
23 # Show the plot
24 plt.tight_layout()
25 plt.show()
26

```

شکل ۱۷: کد رسم کننده نمودارهای جعبه‌ای



شکل ۱۸: نمودارهای جعبه‌ای

سبیل‌های بلند محدوده‌ای از مقادیر بزرگتر را نشان می‌دهد، در حالی که کادر نشان می‌دهد که قسمت عمده داده‌ها کجا قرار دارد. این نشان می‌دهد که در حالی که اکثر املاک دارای محدوده کمتری از اتاق‌ها، اتاق‌خواب‌ها، جمعیت و اندازه‌های خانوار هستند، اما نقاط پرت قابل توجهی با مقادیر بسیار بالا وجود دارد.

این الگو اغلب نشان می‌دهد که چند ملک بسیار بزرگ یا مناطق پر جمعیت وجود دارد، اما اکثریت خانه‌ها، اتاق‌های کمتر و جمعیت/خانوارهای کوچک‌تری دارند.

:Median Income

درآمد متوسط دارای چولگی به راست است اما چولگی کمتری نسبت به کل اتاق ها یا اتاق خواب ها دارد. مقادیر پر کمتری وجود دارد که نشان دهنده توزیع یکنواخت درآمد در میان مجموعه داده است.

Median House Value

توزیع ارزش خانه دارای دامنه وسیعی است، با این جعبه که بسیار گسترده است، که نشان می دهد تغییرات قابل توجهی در قیمت خانه وجود دارد. میانه به چارک پایین نزدیکتر است، که نشان می دهد خانه های بیشتری کمتر از میانه قیمت گذاری شده اند. سبیل بلند سمت راست نشان می دهد که خانه هایی با ارزش به طور قابل توجهی بالاتر از میانه وجود دارد که نشان دهنده وجود املاک لوکس یا با ارزش بالا است.

۴-۲-۱ رسم و تفسیر ماتریس همبستگی

برای رسم ماتریس، ابتدا باید داده های دسته ای را one hot encode کرد. برای این منظور از متده استفاده می کنیم. سپس ستون هدف و ستون ocean_proximity را حذف می کنیم و ابتدا ستون های انکود شده و سپس ستون هدف را به دیتا فریم می چسبانیم. برای این منظور ازتابع pd.concat() استفاده از متد corr() ماتریس همبستگی را ایجاد می کنیم. شکل (۱۹) نحوه انجام این کار را نمایش می دهد.

```
1 # One-hot encode the 'ocean_proximity' column
2 one_hot_encoded = pd.get_dummies(df['ocean_proximity'], prefix='ocean_proximity', dtype=int)
3
4 # Concatenate the one-hot encoded columns with the original DataFrame
5 df_encoded = pd.concat([df.drop(['ocean_proximity', "median_house_value"], axis=1), one_hot_encoded
6 , df.loc[:, "median_house_value"]], axis=1)
7
8 # Calculate the correlation matrix
9 correlation_matrix = df_encoded.corr()
```

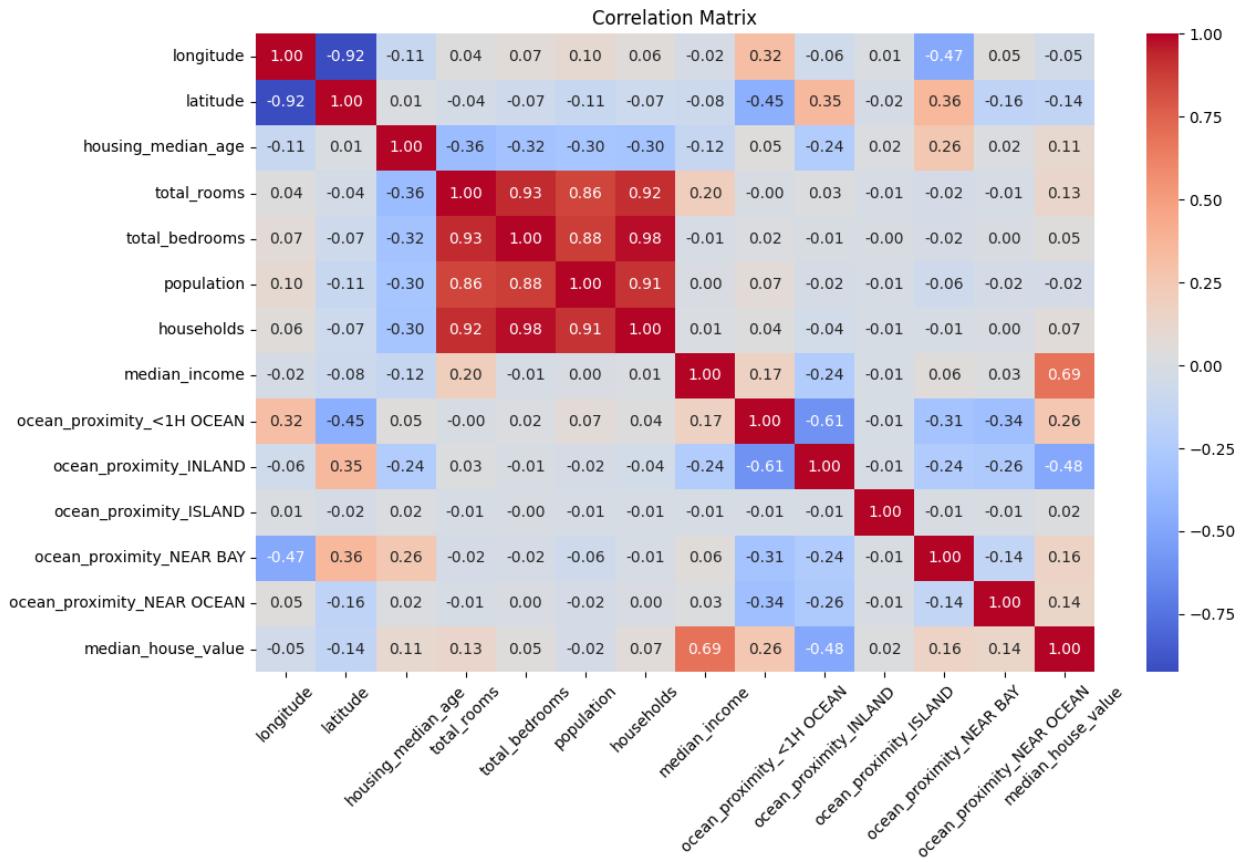
شکل ۱۹: آماده سازی دیتا فریم برای محاسبه ماتریس همبستگی

سپس با استفاده از کتابخانه seaborn و تابع heatmap() ماتریس ایجاد شده را رسم می کنیم. شکل (۲۰) نحوه رسم این ماتریس را نمایش می دهد.

```
1 # Plot the correlation matrix using a heatmap
2 plt.figure(figsize=(12, 8))
3 heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
4 plt.xticks(rotation=45) # Rotate x-axis labels by 45 degrees
5 plt.title('Correlation Matrix')
6 plt.tight_layout() # Adjust layout parameters to ensure all elements are fully visible
7 plt.show()
```

شکل ۲۰: رسم ماتریس همبستگی

همچنین شکل (۲۱) ماتریس رسم شده را نمایش می دهد.



شکل ۲۱: ماتریس همبستگی رسم شده

تفسیر:

طول و عرض جغرافیایی: یک همبستگی منفی قوی بین طول و عرض جغرافیایی وجود دارد، که نشان می‌دهد در این مجموعه داده جغرافیایی، با رفتن به سمت غرب (با افزایش طول جغرافیایی)، فرد تمایل به رفتن به سمت جنوب (کاهش عرض جغرافیایی) دارد یا بر عکس. میانه سن مسکن: به نظر می‌رسد که با اکثر متغیرهای دیگر همبستگی کمی یا بدون وجود دارد، که نشان می‌دهد سن خانه‌ها رابطه خطی قوی با ویژگی‌هایی مانند اتاق‌ها، اتاق خواب‌ها یا اندازه خانه ندارد.

مجموع اتاق‌ها/اتاق‌های خواب و خانوارها/جمعیت: همبستگی‌های مثبت قوی در این گروه‌ها وجود دارد، که شهودی است زیرا اتاق‌های بیشتر معمولاً با تعداد اتاق خواب‌های بیشتر مرتبط هستند و اندازه جمعیت بزرگ‌تر با تعداد خانوارهای بیشتر مرتبط است. درآمد متوسط و میانه ارزش خانه: یک همبستگی مثبت قابل توجه وجود دارد، که نشان می‌دهد مناطق درآمد متوسط بالاتر با ارزش خانه بالاتر مرتبط هستند، که یک الگوی اقتصادی رایج است.

نردهای اقیانوس: دسته‌های مختلف نردهای اقیانوس درجهات مختلفی از همبستگی با میانگین ارزش خانه دارند. این نشان می‌دهد که نردهایی به اقیانوس ممکن است قیمت مسکن را به طور متفاوت تحت تاثیر قرار دهد، به طوری که برخی از مقوله‌ها دارای همبستگی مثبت و برخی دیگر منفی هستند.

۵-۲-۱-رسم و تفسیر نمودار همبستگی با ستون هدف

برای رسم این نمودار، ابتدا با استفاده از متدها corrwith() همبستگی ستون‌ها با ستون هدف را محاسبه می‌کنیم. سپس این مقادیر را در متغیر sort_values() به طور صعودی مرتب می‌کنیم، و با استفاده ازتابع correlation ذخیره می‌کنیم.

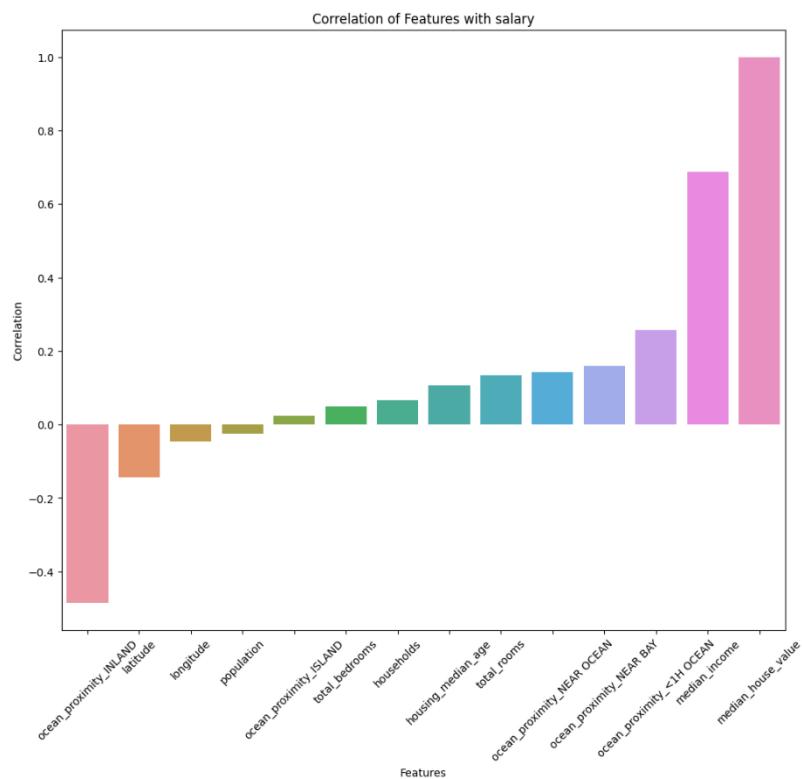
از کتابخانه seaborn از این مقادیر نمودار میله‌ای رسم می‌کنیم. شکل (۲۲) کدی که وظیفه‌ی انجام این عمل را به عهده دارد را نمایش می‌دهد. همچنین شکل (۲۳) خروجی این کد را نمایش می‌دهد.

تفسیر:

همبستگی منفی (نوارهای قرمز و نارنجی):



شکل ۲۲: کد رسم کننده نمودار میزان همبستگی با سطر هدف



شکل ۲۳: نمودار میزان همبستگی با سطر هدف

ویژگی‌های «Latitude» و «ocean_proximity_INLAND» همبستگی منفی با ستون هدف دارند، که نشان می‌دهد با افزایش این مقادیر ویژگی، حقوق تمایل به کاهش دارد.

قوی ترین همبستگی منفی را در بین ویژگی‌های نمایش داده شده دارد.

همبستگی ضعیف تا بدون (نوارهای زرد تا آبی روش):

» «total_bedrooms»، «households»، «ocean_proximity_ISLAND»، «population»، «longitude»، «total_rooms» و «housing_median_age» همبستگی ضعیف و بدون هیچ ارتباطی با ستون هدف نشان می‌دهند.

همبستگی مثبت (نوارهای آبی تا صورتی):

ویژگی‌های «H OCEAN\ocean_proximity_NEAR BAY» همبستگی مثبتی دارند، که نشان می‌دهد با افزایش نزدیکی به این مناطق، میزان ستون هدف نیز افزایش می‌یابد.

«median_house_value» و «median_income» همبستگی مثبت قوی با ستون هدف نشان می‌دهند.

قوی ترین همبستگی‌ها (نوارهای صورتی):

میله 'median_income' بسیار بلند است، که یک همبستگی مثبت قوی را نشان می‌دهد، که به طور بالقوه نشان می‌دهد که این ویژگی می‌تواند پیش بینی خوبی برای ستون هدف باشد.

۱-۳- تشخیص داده‌های پرت

برای تشخیص داده‌های پرت از روش شش سیگما استفاده می‌کنیم. نحوه کارکرد این روش به این صورت است که برای هر ستون میانگین و واریانس محاسبه می‌کنیم و برای هر ستون، هر داده‌ای که کمتر از میانگین منهای سه سیگما و بزرگتر از میانگین به علاوه سه سیگما بود را به عنوان داده پرت شناسایی می‌کنیم. برای مدیریت داده‌های پرت دو رویکرد را با هم مقایسه می‌کنیم. این دو رویکرد عبارت است از حذف داده‌های پرت یا جایگزینی آنها با حد پایین یا حد بالا.

دیتاست حاصل از این دو روش را با الگوریتم رگرسور جنگل تصادفی و ارزیابی مقدار $2R$ مقایسه می‌کنیم.

۱-۳-۱- جایگزینی داده‌ها با حد پایین یا حد بالا

ابتدا از دیتافریم به دست آمده از بخش قبلی، یک کپی در متغیر `cleaned_df` ذخیره می‌کنیم، حالا با استفاده از یک حلقه، برای هر ستون میانگین، واریانس، حد پایین و حد بالا را محاسبه می‌کنیم. سپس مقادیری که بیشتر از حد بالا هستند را با مقدار حد بالا و مقادیری که کمتر از حد پایین هستند را با حد پایین جایگزین می‌کنیم. شکل (۲۴) نحوه انجام این کار را نمایش می‌دهد.

```
1 cleaned_df = df_encoded.copy() # Create a copy of the original DataFrame
2
3 for col in numerical_cols:
4     if col != "salary":
5         lower_limit = cleaned_df[col].mean() - 3 * cleaned_df[col].std()
6         upper_limit = cleaned_df[col].mean() + 3 * cleaned_df[col].std()
7
8     # Filter rows to include only values within the threshold
9     cleaned_df = cleaned_df[(cleaned_df[col] >= lower_limit) & (cleaned_df[col] <= upper_limit)]
10
11 # Reset index to ensure a continuous index after deleting rows
12 cleaned_df.reset_index(drop=True, inplace=True)
```

شکل ۲۴: جایگزینی داده‌های برت با حد بالا و حد پایین

پس از تقسیم داده‌ها به دسته‌های آموزش و تست، داده‌ها را استاندارد سازی می‌کنیم. برای این منظور یک شی از کلاس ColumnTransformer() به نام preprocessor تعریف می‌کنیم. این شی می‌تواند یک یا چند عمل را ستون به ستون روی دیتافریم انجام دهد. سپس یک تاپل که شامل نام عملیات، کلاسی که عملیات را انجام می‌دهد و ستون‌هایی که این عملیات‌ها باید روی آنها انجام شود را به آن پاس می‌دهیم. این کار باعث می‌شود که در صورت اجرای متدها fit() از این شی، عملیات تعریف شده، ستون به ستون روی دیتافریم اجرا شوند. شکل (۲۵) این کلاس و شی ایجاد شده از آن را نمایش می‌دهد.

```
1 # Combine the preprocessing steps using ColumnTransformer
2 preprocessor = ColumnTransformer(
3     transformers=[
4         ('num', StandardScaler(), numerical_cols),
5     ])
```

شکل ۲۵: ایجاد شی از کلاس ColumnTransformer

در ادامه از شی preprocessor متد .fit_transform() را فراخوانی می‌کنیم و دیتاست آموزشی را به آن پاس می‌دهیم. این متد پارامتر دیتابستی که به عنوان آرگومان دریافت کرده را یاد می‌گیرد، سپس با توجه به پارامترهایی که محاسبه کرده، عملیات‌های تعریف شده را انجام می‌دهد. و سپس نتیجه را در همان متغیر قبلی ذخیره می‌کنیم، شکل (۲۶) نحوه عملکرد این کد را نمایش می‌دهد.

سپس برای پیش‌پردازش داده‌های تست، از متد transform() شی استفاده می‌کنیم. دلیل استفاده از این متد این است که این متد با استفاده از پارامترهای یادگرفته شده از داده‌های آموزشی، عملیات را روی داده‌های تستی انجام می‌دهد. توجه شود که اینجا از متد قبلی استفاده نمی‌شود

```
1 # Fit and transform the data using the ColumnTransformer
2 X_train = preprocessor.fit_transform(X_train)
```

شکل ۲۶: پیش‌پردازش داده‌های آموزشی

زیرا فرض بر این است که پارامترهای داده‌های تست را نداریم و باید با استفاده از برآورده که در داده‌های آموزشی انجام دادیم، عملیات‌ها را روی داده‌های تست انجام دهیم. عدم رعایت این مورد، موجب ایجاد نشت داده می‌شود. شکل (۲۷) نحوه پیش‌پردازش داده‌های تست را نمایش می‌دهد.



```
● ● ●  
1 X_valid = preprocessor.transform(X_valid)
```

شکل ۲۷: پیش‌پردازش داده‌های تست

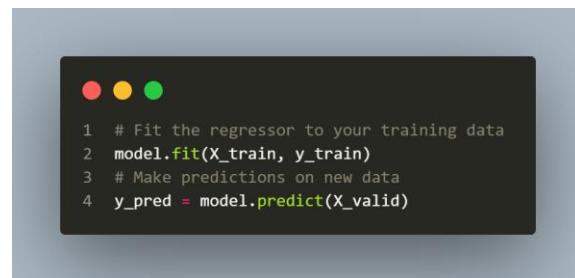
سپس مدل را تعریف می‌کنیم. یک شی از کلاس (`RandomForestRegressor()`) ایجاد می‌کنیم و آن را `model` می‌نامیم. حداقل عمق درخت‌ها را ۱۰، و تعداد برآورده کننده‌ها را برابر ۱۰۰ قرار می‌دهیم. همچنین برای امکان‌پذیری بازتولید نتایج، حالت رندوم را برابر ۴۲ قرار می‌دهیم. شکل (۲۸) تعریف مدل را نمایش می‌دهد.



```
● ● ●  
1 model = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)
```

شکل ۲۸: تعریف مدل

سپس مدل را با استفاده از متدهای `.fit()` و `.predict()` را پاس دادن داده‌های تست به متدهای `.predict()`، پیش‌بینی مدل را در متغیر `y_pred` ذخیره می‌کنیم. شکل (۲۹) نحوه انجام این دو عملیات را نمایش می‌دهد.



```
● ● ●  
1 # Fit the regressor to your training data  
2 model.fit(X_train, y_train)  
3 # Make predictions on new data  
4 y_pred = model.predict(X_valid)
```

شکل ۲۹: آموزش و تست مدل

سپس مقدار `R2` را محاسبه می‌کنیم. شکل (۳۰) نحوه انجام این کار با تابع `r_score()` را نمایش می‌دهد.

```

1 r2 = r2_score(y_valid, y_pred)
2 print("R2 Score:", r2)

```

شکل ۳۰: محاسبه و نمایش ۲R

با روش جایگزینی مقادیر پرت، مقدار ۲R برابر ۰.۷۷۷ می‌شود. حال روش حذف داده‌های پرت را ارزیابی می‌کنیم.

۱-۳-۲- حذف داده‌های پرت

همانند بخش قبلی روی یک کپی از دیتابست بخش قبلی کار خود را انجام می‌دهیم. سپس داده‌ای که به به عنوان پرت شناخته می‌شوند را حذف می‌کنیم. شکل (۳۱) نحوه انجام این کار را نمایش می‌دهد.

```

1 cleaned_df = df_encoded.copy() # Create a copy of the original DataFrame
2
3 for col in numerical_cols:
4     if col != "salary":
5         lower_limit = cleaned_df[col].mean() - 3 * cleaned_df[col].std()
6         upper_limit = cleaned_df[col].mean() + 3 * cleaned_df[col].std()
7
8         # Filter rows to include only values within the threshold
9         cleaned_df = cleaned_df[(cleaned_df[col] >= lower_limit) & (cleaned_df[col] <= upper_limit)]
10
11 # Reset index to ensure a continuous index after deleting rows
12 cleaned_df.reset_index(drop=True, inplace=True)

```

شکل ۳۱: حذف داده‌های پرت

سپس تمامی مراحل را مشابه بخش قبلی انجام می‌دهیم و میزان ۲R را محاسبه می‌کنیم. در این روش، میزان ۲R برابر ۱.۷۵۰ است. بنابراین برای مدیریت داده‌های پرت، از روش حذف آنها استفاده می‌کنیم.

۱-۴- پر کردن مقادیر خالی

برای پر کردن مقادیر خالی دو روش را با هم مقایسه می‌کنیم. اولین روش پر کردن مقادیر با استفاده از KNN است و روش دوم استفاده از میانگین است. دیتابست حاصل از این دو روش را با الگوریتم رگرسور جنگل تصادفی و ارزیابی مقدار ۲R مقایسه می‌کنیم.

۱-۴-۱- تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با KNN

ابتدا داده‌های تمیز شده که در دیتابریم cleaned_df قرار گرفته‌اند را به دو دسته آموزش و تست تقسیم می‌کنیم. در ادامه برای انجام راحت پیش‌پردازش یک شی از کلاس پایپ‌لاین تعریف می‌کنیم. این شی دسته‌ای از عملیات‌ها را به ترتیب روی هدف انجام می‌دهد. شی تعریف شده

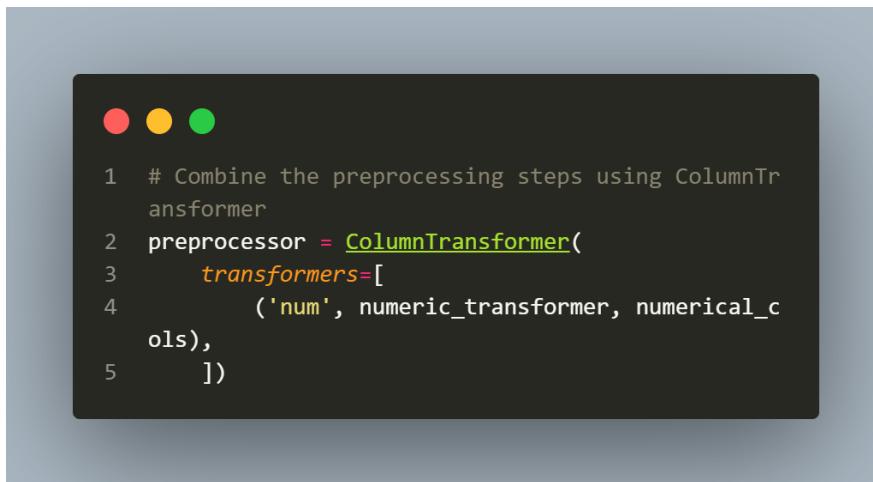
در این قسمت، ابتدا مقادیر خالی را با استفاده از تابع `KNNImputer()` پر می‌کند و سپس داده‌ها را استاندارد سازی می‌کند. شکل (۳۲) کد مربوط به این شی را نمایش می‌دهد.



```
1 numeric_transformer = Pipeline(steps=[  
2     ('imputer', KNNImputer(n_neighbors=100)),  
3     ('scaler', StandardScaler())  
4 ])
```

شکل ۳۲: ایجاد شی از کلاس پایپ‌لاین

سپس همانند بخش قبلی یک شی از کلاس `ColumnTransformer()` ایجاد می‌کنیم و در عضو سوم، شی ساخته شده از پایپ‌لاین را قرار می‌دهیم. شکل (۳۳) ترانسفورمر ایجاد شده را نمایش می‌دهد.



```
1 # Combine the preprocessing steps using ColumnTransformer  
2 preprocessor = ColumnTransformer(  
3     transformers=[  
4         ('num', numeric_transformer, numerical_c  
ols),  
5     ])
```

شکل ۳۳: ایجاد شی از کلاس `ColumnTransformer`

سپس همانند بخش قبلی داده‌های آموزش و تست را پیش‌پردازش می‌کنیم، و مدل را آموزش می‌دهیم. سپس پیش‌بینی را انجام می‌دهیم و مقدار $2R$ را محاسبه می‌کنیم. در این روش، $2R$ برابر با 0.7694 می‌شود.

۴-۴-۱-تعريف شی پیش‌پردازش برای پر کردن مقادیر خالی با میانه حال شی تعریف شده برای پیش‌پردازش را به نحوی تغییر می‌دهیم که مقادیر خالی را با میانه پر کند. برای این منظور، در شی پایپ‌لاین، به جای `KNNImputer()` کلاس `SimpleImputer()` با آرگومان `strategy = 'median'` را پاس می‌دهیم. شکل (۳۴) پایپ‌لاین به روز شده را نمایش می‌دهد.

```

1 numeric_transformer = Pipeline(steps=[
2     ('imputer', SimpleImputer(strategy="median")),
3     ('scaler', StandardScaler())
4 ])

```

شکل ۳۴: پایپلاین به روز شده

سایر مراحل را مشابه بخش قبلی انجام می‌دهیم. و مدل را آموزش و تمرین می‌دهیم و مقدار $2R$ را محاسبه می‌کنیم. در این حالت مقدار $2R = 7694$ خواهد شد. هر دو روش دقیقاً یک مقدار $2R$ را می‌دهند. بنابراین تفاوتی بین انتخاب این دو روش وجود ندارد. از آنجایی که روش پر کردن داده‌ها با میانه به مراتب راحت‌تر است، مقادیر خالی را با میانه پر می‌کنیم.

در ادامه به انتخاب ویژگی می‌پردازیم.

۱-۵-انتخاب ویژگی

برای انتخاب ویژگی از دو روش استفاده می‌کنیم. روش اول استفاده از PCA و روش دوم استفاده از Mutual Information است. سپس این دو روش را همانند روشهایی که در دو قسمت قبلی استفاده شد، با هم مقایسه می‌کنیم.

۱-۱-انتخاب ویژگی با استفاده از روش PCA

ابتدا دیتابست به دست آمده از بخش قبلی را به دو بخش آموزش و تست تقسیم می‌کنیم. سپس یک پایپلاین ایجاد می‌کنیم که در آن به ترتیب داده‌ها استانداردسازی می‌شوند، سپس ۹ مولفه اساسی انتخاب می‌شوند و روی مولفه‌های انتخاب شده، یک الگوریتم رندوم فارست با پارامترهای بخش‌های قبلی آموزش داده می‌شود. شکل (۳۵) این پایپلاین را نمایش می‌دهد.

```

1 # Define the pipeline with PCA and RandomForest
2 pipeline = Pipeline([
3     ('scaler', StandardScaler()),      # Standardize features
4     ('pca', PCA(n_components=9)),      # Perform PCA for dimensionality reduction
5     ('rf', RandomForestRegressor(n_estimators=100, random_state=0)) # RandomForest Classifier
6 ])

```

شکل ۳۵: پایپلاین ایجاد شده برای PCA

سپس با استفاده از متدهای fit() و predict() عملیات‌های گفته شده را روی دیتابست آموزشی اجرا می‌کنیم شکل (۳۶) استفاده از این متدهای نمایش می‌دهد.

سپس با استفاده از متدهای fit() و predict() و پاس دادن داده‌های تست، عملیات‌های گفته شده را روی داده‌های تست انجام می‌دهیم و نتایج پیش‌بینی را ذخیره می‌کنیم. سپس با مقایسه پیش‌بینی و مقادیر حقیقی، مقدار $2R$ را محاسبه می‌کنیم. در این روش مقدار $2R = 7694$ می‌شود.



```

1 # Fit the pipeline on the training data
2 pipeline.fit(X_train, y_train)

```

شکل ۳۶: اجرای پایپلاین روی داده‌های آموزشی

۱-۵-۲- انتخاب ویژگی با روش Mutual Information

ابتدا داده‌های آموزشی و تستی را استاندارد سازی می‌کنیم. سپس با استفاده از کلاس SelectKBest() و آرگومان‌های $k=9$ و score_func=mutual_info_regression یک شی می‌سازیم که متدهای fit() و get_support() را در لیستی به نام selected_features_indices ذخیره می‌کند. سپس با استفاده از fit() آن را در لیستی به نام X_train و y_train بروزرسانی می‌کنیم و فقط ستون‌های حاضر در این لیست را نگه می‌داریم. شکل (۳۷) نحوه انجام این کار را نمایش می‌دهد.



```

1 # Standardize the features
2 scaler = StandardScaler()
3 X_train = scaler.fit_transform(X_train)
4 X_valid = scaler.transform(X_valid)
5
6 # Perform mutual information feature selection
7 selector = SelectKBest(score_func=mutual_info_regression, k=9)
8 selector.fit(X_train, y_train)
9
10 # Get the selected features
11 selected_features_indices = selector.get_support(indices=True)
12 X_train = X_train[:, selected_features_indices]
13 X_valid = X_valid[:, selected_features_indices]

```

شکل ۳۷: انتخاب ویژگی برتر با استفاده از Mutual Information

سپس همانند بخش‌های قبلی یک مدل جنگل تصادفی روی داده‌ها آموزش می‌دهیم و تست می‌کنیم. در این روش R^2 به دست آمده برابر ۰.۷۵۲ است. در نتیجه برای انتخاب ویژگی از روش Mutual Information استفاده می‌کنیم.

حال دیتابستی که تا این مرحله پیش‌پردازش شده است را مجدداً در متغیر df ذخیره می‌کنیم.

۱-۶- پیدا کردن درجه مناسب رگرسیون

در این بخش به بررسی این موضوع می‌پردازیم که کدام درجه از رگرسیون نتیجه‌ی بهتری حاصل می‌کند. به این منظور از Cross Validation استفاده می‌کنیم.

ابتدا دیتافریم را به دو دسته آموزشی و تستی تقسیم می‌کنیم. سپس لیستی از درجات درست می‌کنیم. درجات از یک تا پنج خواهد بود. از آنجایی که زمان آموزش مدل به طور نمایی بالا می‌رود، آموزش مدل از درجه شش و بالاتر نیازمند صرف زمان بسیار زیاد است و نتیجه به دست آمده بهبود نخواهد داشت.

برای ارزیابی از روش MSE استفاده می‌کنیم. برای اینکه امکان رسم نمودار MSE بر اساس درجات را داشته باشیم، یک لیست خالی به نام mse_values_list ایجاد می‌کنیم. سپس یک حلقه ایجاد می‌کنیم که به ازای هر یک از درجات به ترتیب این موارد را انجام می‌دهد:

۱- ابتدا یک پایپ‌لاین ایجاد می‌کند که این پایپ‌لاین به ترتیب داده‌ها را استاندارد می‌کند، سپس درجه مربوطه که توسط حلقه تعریف می‌شود را می‌سازد و مدل رگرسیون را آموزش می‌دهد.

۲- یک لیست خالی تعریف می‌شود که مسئول نگهداری مقدار MSE به دست آمده از هر بخش است.

۳- از کلاس KFold یک شی تعریف می‌شود. این شی دیتابست آموزشی را برابر می‌زند و به پنج بخش تقسیم می‌کند. سپس برای هر بخش، دیتابی آموزشی با استفاده از پایپ‌لاین تعریف شده پیش‌پردازش می‌شود و سپس مدل رگرسیون روی آن آموزش می‌بیند. در ادامه مجدداً با استفاده از پایپ‌لاین، داده‌های تست پیش‌پردازش می‌شوند و پیش‌بینی انجام می‌شود. و در نهایت MSE محاسبه می‌شود و در لیست ایجاد شده ذخیره می‌شود.

۴- این عمل روی تمامی بخش‌ها انجام شده و حلقه به پایان می‌رسد. MSE میانگین این بخش‌ها محاسبه شده و در لیست mse_values_list قرار می‌گیرد.

۵- تمامی عملیات‌های ۱ تا ۴ برای درجه بعدی تکرار می‌شود.

شکل (۳۹) نحوه انجام این عملیات را نمایش می‌دهد.

سپس از مقادیر به دست آمده نمودار رسم می‌کنیم. برای نمایش بهتر مقادیر MSE، محور y را به صورت لگاریتمی رسم می‌کنیم. شکل (۳۸) کدی که وظیفه رسم این نمودار را دارد را نمایش می‌دهد.

```
1 # Plot MSE vs degrees
2 plt.plot(degrees, mse_values_list, marker='o')
3 plt.title('Mean Squared Error vs. Polynomial Degree')
4 plt.xlabel('Polynomial Degree')
5 plt.ylabel('Mean Squared Error')
6 plt.yscale('log')
7 plt.grid(True)
8 plt.show()
```

شکل (۳۸): رسم نمودار از میانگین MSE هر درجه

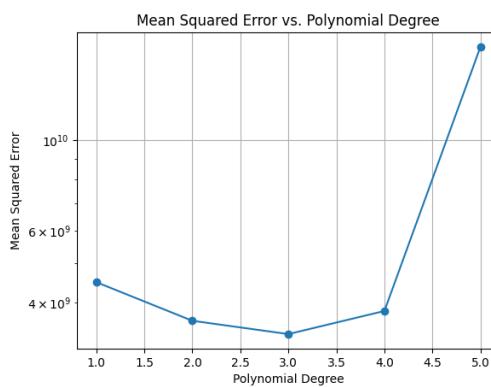
```

1
2 # Define the list to store MSE values
3 mse_values_list = []
4
5 for degree in degrees:
6     # Define the pipeline for each degree
7     pipeline = Pipeline([
8         ('scaler', StandardScaler()), # Standardize the features
9         ('poly_features', PolynomialFeatures(degree=degree)), # Create polynomial features
10        ('regression', linearRegression()) # Linear regression model
11    ])
12
13    mse_values = []
14
15    # Perform k-fold cross-validation
16    kf = KFold(n_splits=5, shuffle=True, random_state=42)
17
18    for train_index, val_index in kf.split(X_train):
19        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
20        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
21
22        # Fit the pipeline on the training data
23        pipeline.fit(X_train_fold, y_train_fold)
24
25        # Predict on validation data
26        y_val_pred = pipeline.predict(X_val_fold)
27
28        # Calculate MSE and store it
29        mse_fold = mean_squared_error(y_val_fold, y_val_pred)
30        mse_values.append(mse_fold)
31
32    # Calculate average MSE for this degree across all folds
33    average_mse = np.mean(mse_values)
34    mse_values_list.append(average_mse)
35    print(f"Mean squared error for degree {degree}: {average_mse}")
36

```

شکل ۳۹: ارزیابی درجه‌های رگرسیون با استفاده از CV

همچنین شکل (۴۰) نمودار رسم شده را نمایش می‌دهد.



شکل ۴۰: نمودار MSE به ازای درجه

با توجه به نمودار، مشخص است که درجه سه کمترین MSE را دارد. در نتیجه مدل را با درجه سه آموزش می‌دهیم. برای آموزش و تست مدل از همان پایپلاین ایجاد شده در حلقه استفاده می‌کنیم. شکل (۴۱) این پایپلاین را نمایش می‌دهد.

```

1 pipeline = Pipeline([
2     ('scaler', StandardScaler()), # Standardize the features
3     ('poly_features', PolynomialFeatures(degree=3)), # Create polynomial features
4     ('regression', LinearRegression()) # Linear regression model
5 ])

```

شکل ۴۱: پایپلاین پیش‌پردازش و آموزش رگرسیون

سپس همانند بخش‌های قبلی R^2 و MSE را محاسبه می‌کنیم. این مقادیر به ترتیب ۰.۹۲۷ و ۵۲۸۹۰۲.۳۲۲۸۵۴۰۹۲۷ به دست می‌آیند.

۱-۷- امتحان مجدد مدل، با استفاده از دو ویژگی جدید
ابتدا ویژگی Population_per_household را بررسی می‌کنیم. برای ایجاد این ویژگی دو ستون متناظر را بر هم تقسیم می‌کنیم و در ستونی با اسم همین ویژگی قرار می‌دهیم و این دو ویژگی را حذف می‌کنیم. شکل (۴۲) نحوه انجام این کار را نمایش می‌دهد.

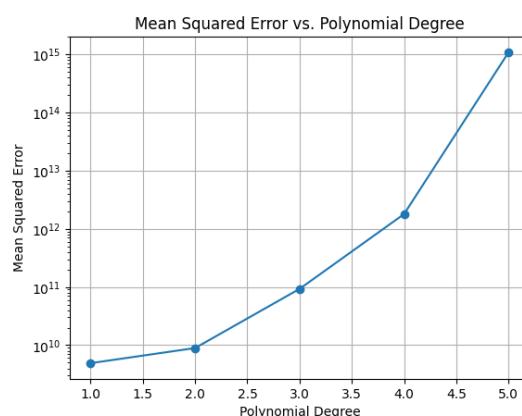
```

1 df['Population_per_household'] = df['population'] / df['households']
2 df.drop(['population', 'households'], axis=1, inplace=True)

```

شکل ۴۲: ایجاد ویژگی جدید با دو ویژگی قدیمی

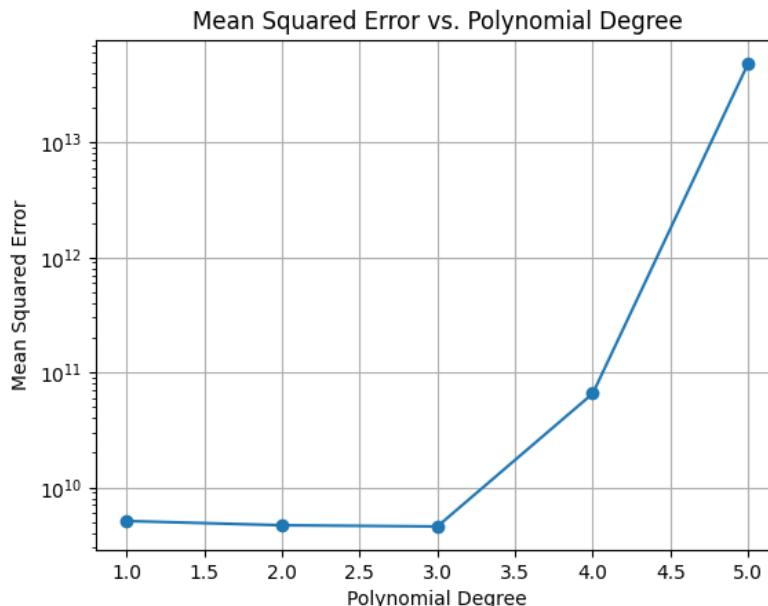
سپس همه مراحل را مانند قسمت قبل تکرار می‌کنیم. نمودار درجه برای این حالت، مانند شکل (۴۳) خواهد بود.



شکل ۴۳: نمودار MSE بر درجه برای حالت جدید دیتابست

سپس با همان روش قبلی یک رگرسیون خطی آموزش و تست و ارزیابی می‌کنیم. در این حالت مقدار MSE و R^2 به ترتیب برابر 534403.48316221435 و 0.4040 خواهد شد.

مشاهده می‌شود که MSE افزایش داشته ولی R^2 کاهش یافته است. سپس ویژگی Rooms_per_household را بررسی می‌کنیم. نحوه ایجاد این ویژگی کاملاً مانند ویژگی قبلی است. در دیتاست اصلی این ویژگی را ایجاد می‌کنیم و سپس نمودار MSE بر اساس درجات را ایجاد می‌کنیم که شکل (۴۴) آن را نمایش می‌دهد.



شکل ۴۴: نمودار MSE براساس درجه در حالت داشتن ویژگی Rooms_per_household

با توجه به شکل مجدداً درجه سوم بهترین عملکرد را دارد ولی عملکرد درجه دو و درجه یک هم بهبود قابل توجهی داشته در حدی که اختلاف این دو درجه با درجه سه بسیار کم است. سپس با همان روش قبلی یک رگرسیون خطی آموزش و تست و ارزیابی می‌کنیم. در این حالت مقدار MSE و R^2 به ترتیب برابر 970041.4419074284 و 0.6430 خواهد بود.

۲-پاسخ سوال دوم

ابتدا دیتاست را لود می‌کنیم. سپس با استفاده از متدهای info() و unique() داده‌های دیتاست را بررسی می‌کنیم. برخی از ستون‌های دیتاست، داده‌های گمشده دارند. همچنین همه ستون‌ها دارای دیتابایپ int یا float هستند حال از متدهای unique() برای بررسی تعداد مقادیر یکتا در هر ستون استفاده می‌کنیم. مشاهده می‌شود که برخی از ستون‌ها فقط دو مقدار یکتا دارند که نشان‌دهنده‌ی دسته‌ای بودن این ستون‌ها است. همچنین ستون education فقط چهار مقدار یکتا دارد که نشان‌می‌دهند یک ستون دسته‌ای با کاردینالیتی چهار است. شکل (۴۵) خروجی متدهای unique() را نمایش می‌دهد.

male	2
age	39
education	4
currentSmoker	2
cigsPerDay	33
BPMeds	2
prevalentStroke	2
prevalentHyp	2
diabetes	2
totChol	248
sysBP	234
diaBP	146
BMI	1364
heartRate	73
glucose	143
TenYearCHD	2
dtype:	int64

شکل (۴۵): خروجی متدهای unique()

حال به بررسی دقیق تعداد مقادیر خالی در هر دسته می‌پردازیم. برای این منظور از method chaining استفاده می‌کنیم. به این صورت که روی متغیر df دو متدهای isnull() و sum() را اجرا می‌کنیم. شکل (۴۶) این method chaining را نمایش می‌دهد.



شکل (۴۶): استفاده از method chaining

خروجی این عملیات در شکل (۴۷) نمایش داده شده.

سپس به بررسی این مورد می‌پردازیم که آیا در سطح‌ها مقادیری مانند وجود دارد یا خیر. همچنین بررسی می‌کنیم که آیا در ستون‌های غیر مجاز، مقدار صفر وجود دارد یا خیر. مثلاً در ستونی BMI مقدار صفر یک مقدار غیر مجاز است زیرا بی‌ام‌آی برابر با صفر نداریم. با بررسی این دو مورد متوجه می‌شویم که ستون‌ها مقادیر غیرمجاز ندارند.

سپس با متدهای drop_duplicates() و dropna() مقادیر تکراری را حذف می‌کنیم.

در مرحله بعدی نام ستون‌های عددی و ستون‌های دسته‌ای را در دو لیست جدا قرار می‌دهیم. برخلاف سوال قبلی نمی‌توانیم از دیتابایپ استفاده کنیم چون که داده‌های دسته‌ای به شکل رشته نیستند و به صورت عددی هستند. شکل (۴۸) این تقسیم بندی را نمایش می‌دهد.

```

male          0
age           0
education     105
currentSmoker 0
cigsPerDay    29
BPMeds        53
prevalentStroke 0
prevalentHyp   0
diabetes      0
totChol       50
sysBP         0
diaBP         0
BMI           19
heartRate     1
glucose       388
TenYearCHD    0
dtype: int64

```

شکل ۴۷: تعداد مقادیر خالی در هر ستون



```

1 numerical_cols = ["age", "cigsPerDay", "totChol", "sysBP", "diaBP", "BMI", "heartRate",
                    "glucose", ]
2 categorical_cols = ["male", "education", "currentSmoker", "BPMeds", "prevalentStroke",
                      "prevalentHyp", "diabetes"]

```

شکل ۴۸: ذخیره نام ستون‌ها در دو لیست مجزا

۱-۲-پر کردن مقادیر خالی

ابتدا دیتاست را به دو دسته آموزشی و تستی تقسیم می‌کنیم. برای پر کردن مقادیر خالی ستون‌های عددی، از دو روش استفاده می‌کنیم. پر کردن مقادیر خالی با استفاده از KNN و پر کردن مقادیر با استفاده از میانه. برای پر کردن مقادیر خالی ستون‌های دسته‌ای از مد استفاده می‌کنیم. به این منظور دو شی از کلاس Pipeline ایجاد می‌کنیم که کارکرد آن، اجرای یک یا چند عملیات به طور متوالی است. پایپلاین ایجاد شده برای داده‌های عددی به این صورت است که ابتدا مقادیر را استاندارد سازی می‌کنیم، سپس با استفاده از KNN مقادیر خالی را پر کنیم. شکل (۴۹) این پایپلاین را نمایش می‌دهد.



```

1 numeric_transformer = Pipeline(steps=[
2     ('scaler', StandardScaler()),
3     ('imputer', KNNImputer(n_neighbors=100)),
4 ])

```

شکل ۴۹: ایجاد پایپلاین برای داده‌های عددی

پایپلاین مقادیر دسته‌ای به این صورت است که با استفاده از کلاس SimpleImputer و آرگومان strategy = ‘most_frequent’ مقادیر خالی را با استفاده از مد پر می‌کنیم و سپس داده‌ها را استانداردسازی می‌کنیم. شکل (۵۰) پایپلاین داده‌های دسته‌ای را نمایش می‌دهد.

```
1 numeric_transformer = Pipeline(steps=[  
2     ('scaler', StandardScaler()),  
3     ('imputer', KNNImputer(n_neighbors=100)),  
4 ])
```

شکل ۵۰: پایپلاین برای مقادیر دسته‌ای

سپس یک شی از کلاس ColumnTransformer ایجاد می‌کنیم، و به آرگومان transformers لیستی شامل دو تاپل را پاس می‌دهیم. هر تاپل به ترتیب شامل یک اسم، شی پایپلاین و لیست ستون‌های متناظر با پایپلاین است. این شی عملیات‌های تعریف شده در پایپلاین‌های بالا را ستون به ستون روی لیست ستون‌های داده شده اجرا می‌کند. شکل (۵۱) این شی را نمایش می‌دهد.

```
1 # Combine the preprocessing steps using ColumnTransformer  
2 preprocessor = ColumnTransformer(  
3     transformers=[  
4         ('num', numeric_transformer, numerical_cols),  
5         ('cat', categoric_transformer, categorical_cols),  
6     ])
```

شکل ۵۱: شی ساخته شده از ColumnTransformer

حال از یکی شی دیگر از کلاس پایپلاین استفاده می‌کنیم تا شی preprocessor و مدل جنگل تصادفی را به آن می‌دهیم. مزیت استفاده از این روش این است که می‌توان چندین عملیات را به طور زنجیره‌وار برای اجرا شدن تعریف کنیم و از اجرای بی‌نقص آنها اطمینان حاصل کنیم. همچنین کد نوشته شده با این روش بسیار خواناست و همچنین عوض کردن آن برای امتحان و اجرای روش‌های مختلف به راحتی صورت می‌گیرد. شکل (۵۲) پایپلاین نهایی را نمایش می‌دهد.

```
1 pipeline = Pipeline(  
2     steps=[  
3         ('preprocessor' , preprocessor),  
4         ("model" , RandomForestClassifier(n_jobs = -1 , random_state=42))  
5     ]  
6 )
```

شکل ۵۲: پایپلاین نهایی

این پایپلاین به شرح زیر عمل می‌کند:

۱- مقادیر خالی ستون‌های عددی را با KNN پر می‌کند و سپس داده‌ها را استاندارد می‌کند.

۲- مقادیر خالی ستون‌های دسته‌ای را با مد پر می‌کند و داده‌ها را استاندارد می‌کند.

۳- بسته به متدهایی که در ادامه استفاده می‌شود، مدل را آموزش می‌دهد یا تست می‌کند.

برای ارزیابی روش انتخاب شده، از Cross Validation استفاده می‌کنیم. برای این منظور از کلاس cross_val_score یک شی می‌سازیم. به این شی، پایپلاین نهایی و داده‌های آموزشی، روش ارزیابی و تعداد بخش‌ها را پاس می‌دهیم. این بخش از کد، داده‌ها را به پنج دسته تقسیم می‌کند، عملیات‌های گفته شده را روی داده‌های آموزش و ارزیابی انجام می‌دهد و مقدار ۲R را محاسبه می‌کند.

سپس میانگین و واریانس مقادیر ۲R را محاسبه می‌کنیم و نمایش می‌دهیم. شکل (۵۳) نحوه کارکرد این بخش را نمایش می‌دهد.

```
1 # Perform cross-validation
2 cv_scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='r2')
3
4 # Calculate mean and standard deviation of R2 scores
5 mean_r2 = cv_scores.mean()
6 std_r2 = cv_scores.std()
7
8 print(f"Mean R2 score: {mean_r2}")
9 print(f"Standard deviation of R2 score: {std_r2}")
```

شکل ۵۳: انجام Cross Validation و محاسبه ۲R

با این روش مقدار ۲R برابر با 0.20 - به دست می‌آید.

حال همین متدهای پر کردن مقادیر با میانه استفاده می‌کنیم. با این تفاوت که در پایپلاین مربوط به پیش‌پردازش داده‌های عددی، به جای کلاس KNNImputer() از کلاس SimpleImputer با آرگومان strategy = 'median' استفاده می‌کنیم. شکل (۵۴) پایپلاین به روز شده را نمایش می‌دهد.

```
1 numeric_transformer = Pipeline(steps=[
2     ('imputer', SimpleImputer(strategy='median')),
3     ('scaler', StandardScaler()),]
4 )
```

شکل ۵۴: پایپلاین به روز شده برای مقادیر عددی

حال تمامی مراحل را مجدداً تکرار می‌کنیم. با روش جدید، مقدار ۲R برابر با 0.21 - می‌شود.

روش KNN برای پر کردن مقادیر گمشده عملکرد بهتری دارد.

۲-۲-استفاده از SelectKBest

این روش چگونه کار می‌کند؟ SelectKBest یک روش انتخاب ویژگی از scikit-learn است که k ویژگی برتر را بر اساس بالاترین امتیاز از یک آزمون آماری بین هر ویژگی و متغیر هدف انتخاب می‌کند. هنگامی که SelectKBest اعمال می‌شود، ابتدا امتیاز هر ویژگی را با استفاده از تابع امتیازدهی ارائه شده، مانند χ^2 ، f_{classif} یا $f_{\text{regression}}$ برای وظایف طبقه بندی، یا mutual_info_classif برای وظایف رگرسیون محاسبه می‌کند. این امتیازات وابستگی بین هر ویژگی و هدف را اندازه گیری می‌کند و به SelectKBest اجازه می‌دهد تا ویژگی‌ها را رتبه بندی کند. پس از رتبه‌بندی، فقط k ویژگی‌های دارای بالاترین امتیاز حفظ می‌شوند و بقیه کنار گذاشته می‌شوند.

برای استفاده از این روش، ابتدا متغیرها و ستون هدف را در دو متغیر X و y ذخیره می‌کنیم. سپس از کلاس SelectKBest() یک شی می‌سازیم. آرگومان score_func را با کلاس χ^2 از کتابخانه سایکیتلرن مقدار دهی می‌کنیم و آرگومان k را برابر ۱۰ قرار می‌دهیم. سپس با استفاده از متد fit_transform() و پاس دادن متغیرها و سطر هدف، ده ویژگی برتر را انتخاب می‌کنیم و نتیجه را در $X_{\text{best_features}}$ ذخیره می‌کنیم. حال با استفاده از متد get_support()، اندیس ستون‌های انتخاب شده را در یک لیست ذخیره می‌کنیم سپس نام ستون‌های انتخاب شده را در لیست selected_features ذخیره می‌کنیم. شکل (۵۵) نحوه انجام این عملیات‌ها را نمایش می‌دهد.



```
● ● ●  
1 # Separate features (X) and target variable (y)  
2 X = df.drop(columns=['TenYearCHD'])  
3 y = df['TenYearCHD']  
4  
5 k_best_selector = SelectKBest(score_func=chi2, k=10) # Select top 10 features  
6 X_best_features = k_best_selector.fit_transform(X, y)  
7  
8 # Get the indices of the selected features  
9 selected_feature_indices = k_best_selector.get_support(indices=True)  
10  
11 # Get the names of the selected features  
12 selected_features = X.columns[selected_feature_indices]
```

شکل ۵۵: انتخاب ویژگی با استفاده از SelectKBest

حال می‌توان از لیست ایجاد شده، به راحتی برای انتخاب ستون‌های دیتافریم استفاده کرد. پس از انتخاب این ستون‌ها به سراغ آموزش مدل KNN می‌رویم.

۳-۲-آموزش KNN

ابتدا دیتاست را به دو دسته آموزشی و تستی تقسیم می‌کنیم. توجه شود از آنجایی که دیتاست imbalance است، در اولین اقدام باید تعداد لیبل‌های صفر و تعداد لیبل‌های یک را در دیتاست آموزشی برابر کرد. برای این منظور از روش SMOTE استفاده می‌کنیم. این روش یکی از روش‌های oversampling است که از لیبلی که تعداد کمتری دارد داده‌های مصنوعی تولید می‌کند تا تعداد هر دو لیبل با هم برابر شود.

برای این منظور از کلاس SMOTE یک شی به نام sm می‌سازیم. سپس با استفاده از متد fit_resample() داده‌های آموزشی را بالанс می‌کنیم. شکل (۵۶) نحوه انجام این کار را نمایش می‌دهد.

```

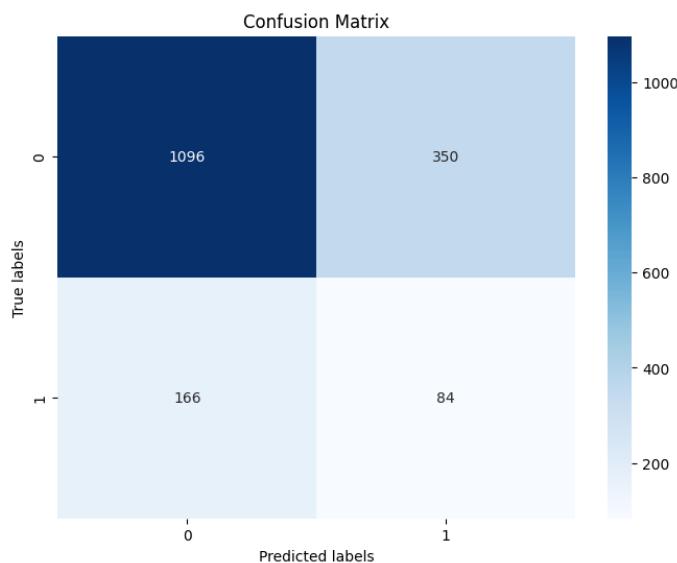
1 sm = SMOTE(random_state=2)
2
3 print("\nClass 1 before Over Sampling --> ", sum(y_train == 1))
4 print("\nClass 0 before Over Sampling --> ", sum(y_train == 0))
5
6 X_train, y_train = sm.fit_resample(X_train, y_train)
7
8 print("\nThe shape of X after Over Sampling -->", X_train.shape)
9 print("\nThe shape of Y after Over Sampling -->", y_train.shape)
10
11 print("\nClass 1 after Over Sampling --> ", sum(y_train == 1))
12 print("\nClass 0 after Over Sampling --> ", sum(y_train == 0))
13 print("\n")

```

شکل ۵۶: Oversample کردن داده‌های آموزشی

حال داده‌ها برای آموزش آماده هستند. برای آموزش داده‌ها همانند بخش‌های قبلی یک پایپ‌لاین به این صورت است که ابتدا داده‌ها را استاندارد سازی می‌کنیم سپس با استفاده از کلاس KNeighborsClassifier() و آرگومان n_neighbors = 3 یک مدل کا نزدیک‌ترین همسایه را آموزش یا تست می‌کنیم. با استفاده از متدها fit() و predict() پاس دادن داده‌های آموزشی، ابتدا داده‌ها را استاندارد سازی می‌کنیم و سپس مدل را آموزش می‌دهیم. همچنانی با متدهای accuracy_score() و score() دقت را محاسبه می‌کنیم. در این حالت، این دو مقدار به ترتیب ۰.۴۲ و ۶۹.۵۷ درصد به دست می‌آید.

سپس با تابع confusion_matrix() ماتریس درهم ریختگی را محاسبه می‌کنیم و با استفاده از تابع heatmap() آن را رسم می‌کنیم. شکل ۵۷ ماتریس درهم ریختگی به دست آمده را نمایش می‌دهد.



شکل ۵۷: ماتریس درهم ریختگی مدل آموزش داده شده

۴-۲ آموزش و تست مدل با فاصله‌های متفاوت

فاصله‌ی پیش‌فرض مدل KNN فاصله اقلیدسی است. حال با فواصل مختلف، آموزش و تست را تکرار می‌کنیم.

۱-۴-۲ فاصله منهتن

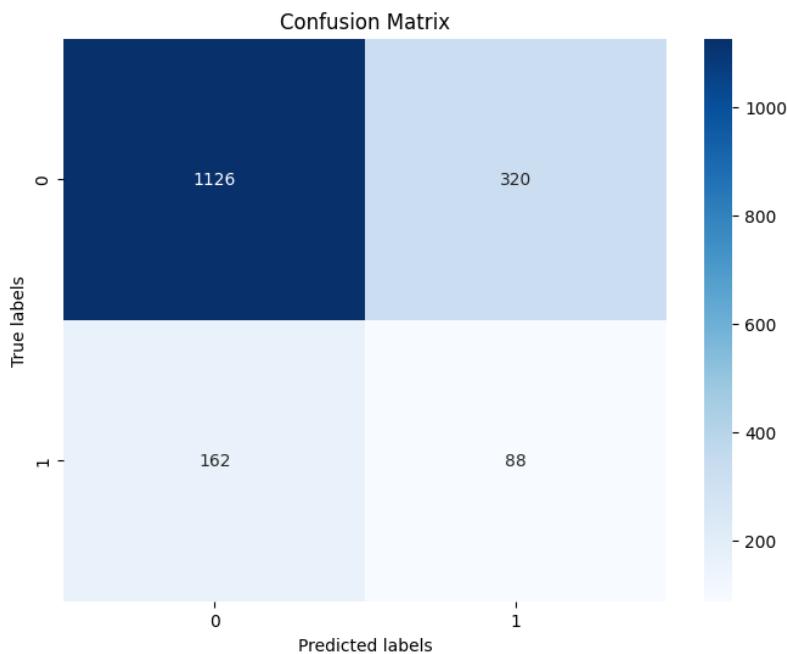
پایپ‌لاین آموزش را به این صورت تغییر می‌دهیم که آرگومان metric را برابر 'manhattan' قرار می‌دهیم. شکل (۵۸) پایپ‌لاین به روز شده را نمایش می‌دهد.



```
1 pipeline = Pipeline(
2     steps=[
3         ('scaler', StandardScaler()),
4         ("model", KNeighborsClassifier(n_neighbors=3, metric='manhattan'))
5     ]
```

شکل ۵۸: پایپ‌لاین آموزش/تست با فاصله منهتن

سپس همانند قبل مراحل آموزش و تست و ارزیابی را تکرار می‌کنیم. در این حالت مقدار $2R$ برابر با ۱.۲۶ و دقت برابر با ۰.۷۱۵۸ به دست می‌آید. مشاهده می‌شود که نتایج بهبود داشته‌اند. همچنین شکل (۵۹) ماتریس درهم‌ریختگی حالت جدید را نمایش می‌دهد.



شکل ۵۹: ماتریس درهم‌ریختگی KNN با فاصله منهتن

۲-۴-۲- فاصله کسینوسی

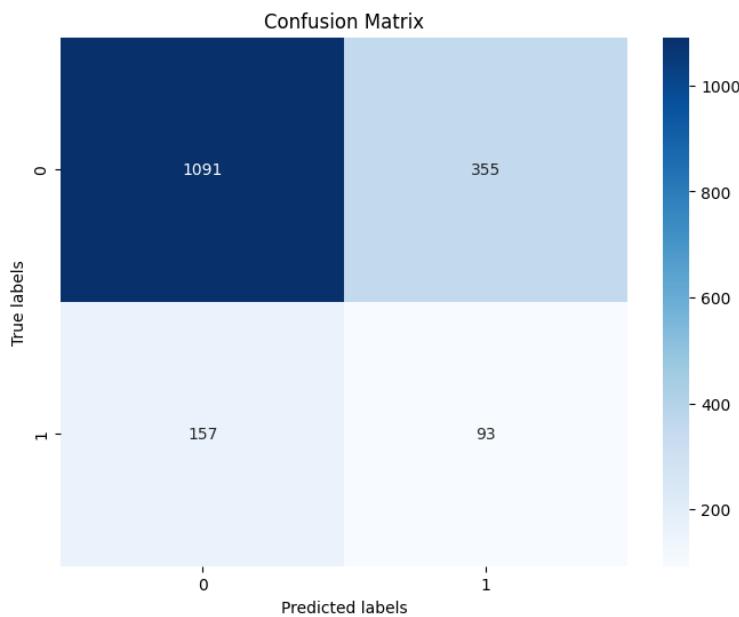
پایپلاین آموزش را به این صورت تغییر می‌دهیم که آرگومان `metric` را برابر 'cosine' قرار می‌دهیم. شکل (۶۰) پایپلاین به روز شده را نمایش می‌دهد.



```
1 pipeline = Pipeline(
2     steps=[
3         ('scaler', StandardScaler()),
4         ("model", KNeighborsClassifier(n_neighbors=3, metric='cosine'))
5     ]
6 )
```

شکل ۶۰: پایپلاین تست و آموزش KNN با فاصله کسینوسی

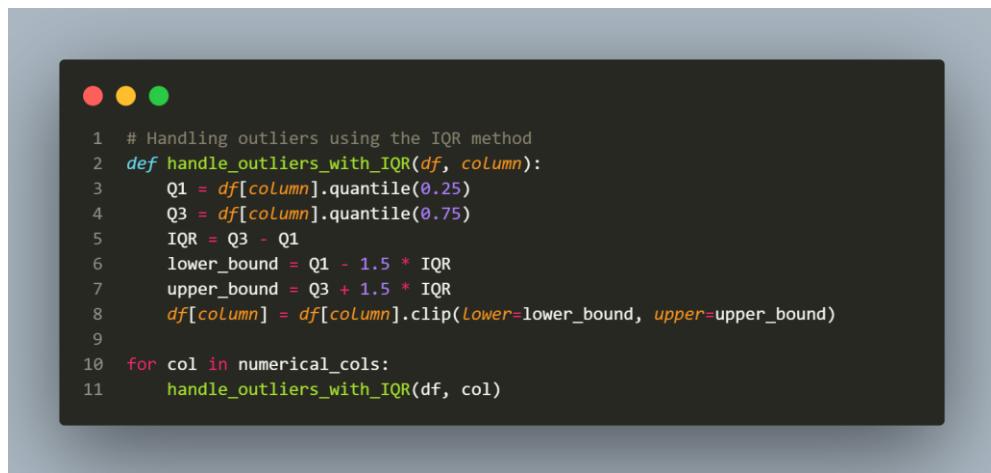
مراحل تست و آموزش را تکرار می‌کنیم. در این حالت $2R$ برابر با 1.4×0.6981 و دقت برابر 0.984 به دست می‌آیند. مشاهده می‌شود که عملکرد فاصله کسینوسی با فاصله اقلیدسی تفاوت چندانی ندارد.



شکل ۶۱: ماتریس درهم‌یختگی مدل KNN با فاصله کسینوسی

۲-۵-۲- انجام پیش‌پردازش بیشتر برای رسیدن به نتایج بهتر

یکی از مراحل پیش‌پردازشی که انجام نشد، پیدا کردن و حذف داده‌های پرت بود. به این منظور. ابتدا برای هر ستون، مقادیر چارک اول و سوم و دامنه میان چارکی را پیدا می‌کنیم، سپس داده‌های بزرگتر از ۱.۵ برابر دامنه به علاوه چارک سوم و داده‌های کوچک‌تر از ۱.۵ برابر دامنه منهای چارک اول هستند را حذف می‌کنیم. برای این منظور از تابع `clip()` و تعریف دو مقدار توضیح داده شده به عنوان حد بالا و حد پایین، تنها داده‌هایی را نگه می‌داریم که بین دو حد تعریف شده هستند. شکل (۶۱) نحوه این انجام را نمایش می‌دهد.



```
 1 # Handling outliers using the IQR method
 2 def handle_outliers_with_IQR(df, column):
 3     Q1 = df[column].quantile(0.25)
 4     Q3 = df[column].quantile(0.75)
 5     IQR = Q3 - Q1
 6     lower_bound = Q1 - 1.5 * IQR
 7     upper_bound = Q3 + 1.5 * IQR
 8     df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)
 9
10 for col in numerical_cols:
11     handle_outliers_with_IQR(df, col)
```

شکل ۶۲ حذف داده‌های پرت

سپس تقسیم بندی داده‌ها به آموزش و آزمایش را انجام می‌دهیم و با استفاده از پایپلاین بخش قبلی، پس از استانداردسازی داده‌ها، یک مدل KNN با تعداد همسایگی ۳ و فاصله منهتن را آموزش می‌دهیم و پس از تست، ارزیابی می‌کنیم.

در این حالت مقدار $2R$ برابر با 0.48 و دقت برابر با 0.8136 به دست می‌آید که به وضوح تاثیر حذف داده‌ها بر عملکرد مدل را نشان می‌دهد.

۳-پاسخ سوال سوم

پس از خواندن دیتابست، ابتدا تعداد رکورد موجود از هر لیبل را نمایش می‌دهیم. شکل (۶۳) خروجی این کار را نمایش می‌دهد.

```
label
ham      500
spam     500
Name: count, dtype: int64
```

شکل ۶۳: تعداد ایمیل‌های با لیبل سالم و اسپم

مشاهده می‌شود که دیتابست بالاتس است. در ادامه لیبل‌های ham را به صفر و spam را به یک تبدیل می‌کنیم.

۱-۳-پیش پردازش داده‌ها

۱-۱-۳-حذف کاراکترهای \n

اولین قدم، حذف کاراکتر \n از رشته‌ی هر ایمیل است. برای این منظور از متدهای replace() استفاده می‌کنیم. کد شکل (۶۴) نحوه انجام این کار را نمایش می‌دهد.

```
● ● ●
1 # Remove the "\n" characters from the text column
2 df['text'] = df['text'].str.replace('\n', ' ')
3
```

شکل ۶۴: حذف کاراکتر \n

۲-۱-۳-حذف کاراکترهای انگلیسی

قدم بعدی حذف کلمات و کاراکترهای انگلیسی است. دلیل انجام این کار این است که زبان ایمیل‌ها فارسی است و وجود کاراکترهای انگلیسی باعث می‌شود که مدل روی کلماتی آموزش ببیند که لزوماً اطلاعاتی برای تصمیم‌گیری مدل ارائه نمی‌دهد. همچنین پس از حذف علائم نگارشی، از مواردی مانند ایمیل یا لینک‌ها، یک رشته‌ی بلند و نه چندان معنادار از کاراکترهای فارسی می‌ماند که مدل را گمراه می‌کند.

برای این منظور از کتابخانه رجکس استفاده می‌کنیم. ابتدا یکتابع را تعریف می‌کنیم که ردیف به ردیف سطرها را دریافت می‌کند و سپس با استفاده از متدهای sub() کاراکترهای انگلیسی را حذف می‌کند و متن تمیز شده را برمی‌گرداند. شکل (۶۵) نحوه انجام این کار را نمایش می‌دهد.

۳-۱-۳-حذف علائم نگارشی

در مرحله بعدی، علائم نگارشی را حذف می‌کنیم. دلیل انجام این کار این است که تکرار یکیسری از علائم مانند نقطه یا اتساین باعث گمراه شدن مدل می‌شود در حالی که حضور این موارد تعیین کننده اسپم بودن ایمیل‌ها نیست. شکل (۶۶) نحوه انجام این کار با متدهای maketrans()

```
1 # Function to remove English words and characters
2 def remove_english(text):
3     # Regex to remove words with English letters and standalone English letters
4     cleaned_text = re.sub(r'\b[a-zA-Z]+\b', '', text) # Removes words composed only of English letters
5     cleaned_text = re.sub(r'[a-zA-Z]', '', cleaned_text) # Removes standalone English letters
6     return cleaned_text
7 df['text'] = df['text'].apply(remove_english)
```

شکل ۶۵: حذف کاراکترهای انگلیسی

```
1 # Function to remove punctuation from text
2 def remove_punctuation(text):
3     # Define a translation table with all punctuation characters mapped to None
4     translator = str.maketrans(' ', ' ', string.punctuation)
5     # Remove punctuation using the translation table
6     return text.translate(translator)
7
8 # Apply the function to the "Content" column
9 df['text'] = df['text'].apply(remove_punctuation)
```

شکل ۶۶: حذف علامت نگارشی

را نمایش می‌دهد. همچنین علامت نگارشی از اتریبیوت string.punctuation فراخوانی خواهد شد.

۴-۱-۳- حذف اعداد

پس از انجام این مراحل، اعداد را از ایمیل‌ها حذف می‌کنیم. وجود اعداد باعث ایجاد نویز در داده‌ها می‌شود. برای حذف اعداد مجدداً همانند قبل از رجکس و تابعی استفاده می‌کنیم که با متده apply() سطر به سطر ایمیل‌ها را دریافت می‌کند و اعداد را از آنها حذف می‌کند.

```
1 # Function to remove numbers from text using regular expressions
2 def remove_numbers(text):
3     # Use regular expression to remove all numbers
4     return re.sub(r'\d+', '', text)
5
6 df['text'] = df['text'].apply(remove_numbers)
```

شکل ۶۷: حذف اعداد با استفاده از رجکس

۵-۱-۳- نرمال کردن متن‌ها

نرمال کردن متن ریشه‌بندی، لغتسازی و حذف کلید واژه، ابعاد داده‌های متن را با جمع کردن کلمات مشابه در یک نمایش مشترک کاهش می‌دهد. این کار فضای ویژگی‌ها را ساده می‌کند و کارایی الگوریتم‌ها را بهبود می‌بخشد. برای نرمال کردن متن از کتابخانه Hazm یک شی از کلاس Normalizer می‌سازیم. سپس روی ستون text متده apply() را اجرا می‌کنیم و یکتابع لامبدا به این متده پاس می‌دهیم که آرگومان آن متن است. سپس متده normalize() را روی متن پاس داده شده اجرا می‌کنیم و نتیجه را بر می‌گردانیم. شکل (۶۸) نحوه انجام نرمال‌سازی را نمایش می‌دهد.



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

شکل ۶۸: نحوه انجام نرمال‌سازی متن

۶-۱-۳- توکن کردن متن

توکن‌سازی متن را به واحدهای کوچک‌تری که می‌توانند کلمات، عبارات یا کاراکترها باشند، تجزیه می‌کند. این جزئیات تجزیه و تحلیل دقیق‌تری از متن را امکان‌پذیر می‌کند و به الگوریتم‌ها اجزاء می‌دهد تا بر روی واحدهای جداگانه کار کنند تا متن. توکن‌ها ساده‌ترین جز سازنده برای ایجاد ویژگی‌ها از متن هستند.

برای توکن کردن متن از کلاس WordTokenizer کتابخانه Hazm استفاده می‌کنیم. دقیقاً مانند قسمت قبلی در متده apply() یکتابع لامبدا ایجاد می‌کنیم و با متده tokenize سطر به سطر متن‌ها را توکن می‌کنیم. لیست توکن‌های هر سطر را در یک ستون جداگانه به نام Tokenized_Content ذخیره می‌کنیم. شکل (۶۹) نحوه انجام این کار را نمایش می‌دهد.



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

شکل ۶۹: توکن‌سازی متن

۷-۱-۳- حذف ایستوازه‌ها (کلمات پالایشی یا stop words)

برای حذف ایستوازه‌ها، ابتدا این واژه‌ها را از فایل PersianStopWords.txt می‌خوانیم و در متغیر stop_words ذخیره می‌کنیم. سپس لیست‌های موجود در ستون Tokenized_Content را با متده apply یک به یک می‌خوانیم. سپس با استفاده از list comprehension توکن‌ها را یک به یک بررسی می‌کنیم که آیا در لیست ایستوازه‌ها موجود هستند یا نه. سپس توکن‌هایی که در ایستوازه‌ها موجود هستند را حذف می‌کنیم. شکل ۷۰ نحوه انجام این کار را نمایش می‌دهد.



```

1 # Read stop words from the text file
2 with open("PersianStopWords.txt", "r", encoding="utf-8") as file:
3     stop_words = set(file.read().splitlines())
4
5 # Define a function to remove stop words
6 def remove_stop_words(tokens):
7     return [word for word in tokens if word not in stop_words]
8
9 # Apply the function to the "Tokenized_Content" column
10 df['Tokenized_Content'] = df['Tokenized_Content'].apply(remove_stop_words)

```

شکل ۷۰: پیدا کردن و حذف ایستوازه‌ها

سپس توکن‌ها را در کنار هم قرار می‌دهیم و متن‌های بلند را مجدداً تشکیل می‌دهیم. (به اصطلاح flatten کردن) شکل (۷۱) نحوه انجام این کار را نمایش می‌دهد.



```

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

```

شکل ۷۱: بازتولید متن‌ها از کلمه‌ها

۸-۱-۳- استفاده از TF-IDF

TF-IDF به هر عبارت در یک سند بر اساس فراوانی آن در سند (TF) و نادر بودن آن در همه اسناد (IDF) وزن اختصاص می‌دهد. این طرح وزن دهی به اولویت بندی عباراتی که هم در یک سند متداول هستند و هم در کل مجموعه منحصر به فرد هستند، کمک می‌کند و آنها را برای کارهای پایین دستی مانند طبقه‌بندی یا خوشه‌بندی متمایزتر و آموزنده‌تر می‌کند.

برای استفاده از TF-IDF از کلاس TfidfVectorizer() یک شی می‌سازیم. سپس با استفاده از متده fit_transform() و پاس دادن ستون متن بردارها را ایجاد می‌کنیم و سپس نتیجه را در یک متغیر دیگر ذخیره می‌کنیم. سپس ماتریس به دست آمده را به دیتا فریم تبدیل می‌کنیم. شکل (۷۲) نحوه انجام این کار را نمایش می‌دهد.

حال دیتابست برای آموزش KNN آماده است. این دیتابست هزار سطر و ۱۸۸۸۶ ستون دارد.

۲-۳-آموزش KNN و پیدا کردن بهترین K

ابتدا دیتابست را به دو دسته آموزشی و آزمایشی تقسیم می‌کنیم. ابتدا داده‌ها را استاندارد سازی می‌کنیم. در مرحله بعدی با استفاده از Oversampling تعداد سطرهای بالیبل صفر و تعداد لیبل‌های یک را با هم برابر می‌کنیم. برای این منظور از روش SMOTE استفاده می‌کنیم. برای این منظور از کلاس SMOTE یک شی می‌سازیم و با پاس دادن داده‌های آموزشی به متده fit_resample() کار oversampling را انجام می‌دهیم.

```

1 # Create a TfidfVectorizer object
2 tfidf_vectorizer = TfidfVectorizer()
3
4 # Fit and transform the 'text' column of the DataFrame
5 tfidf_matrix = tfidf_vectorizer.fit_transform(df['text'])
6
7 # Convert to DataFrame
8 tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
9

```

شکل ۷۲: استفاده از روش TF-IDF

سپس از کلاس KNeighborsClassifier() یک شی می‌سازیم. همچنین یک دیکشنری ایجاد می‌کنیم و پارامترهایی که قرار است روی آنها Grid search انجام بدهیم را به عنوان کلید و مقادیر پارامترها را به عنوان value در دیکشنری قرار می‌دهیم. شکل (۷۳) تعریف شی و این دیکشنری را نمایش می‌دهد.

```

1 # Define the model and parameters
2 knn = KNeighborsClassifier()
3 param_grid = {'n_neighbors': np.arange(1, 21)}

```

شکل ۷۳: تعریف مدل و دیکشنری پارامترها

سپس از کلاس GridSearchCV() یک شی می‌سازیم. آرگومان‌های پاس داده شده به این کلاس، به ترتیب مدل تعریف شده، دیکشنری پارامترها، تعداد بخش‌ها برای CrossValidation و معیار انتخاب شده برای ارزیابی است. تعداد بخش‌ها را پنج قرار می‌دهیم و معیار ارزیابی را دقت قرار می‌دهیم. سپس با متدهای fit() و np.arange() پاس دادن داده‌های آموزشی، Grid search را آغاز می‌کنیم.

شی ساخته شده از GridSearchCV() هر بار پارامتر n_neighbors را تنظیم می‌کند، CV را انجام می‌دهد و در نهایت یک مقدار برای دقت مدل باز می‌گرداند. حال می‌توان از این مقادیر، یک نمودار دقت بر اساس K رسم کرد.

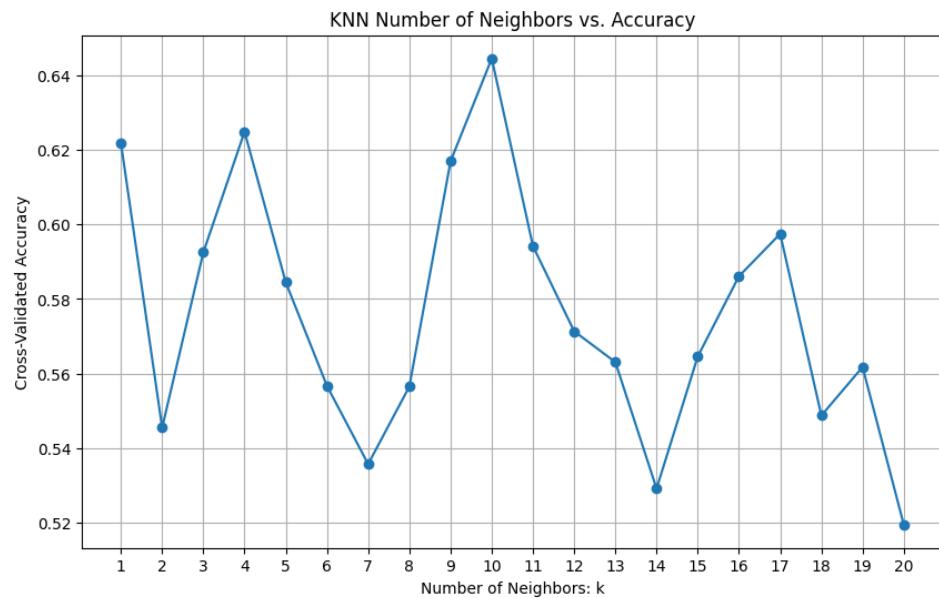
شکل (۷۴) کدی که این نمودار را رسم می‌کند نمایش می‌دهد. همچنین شکل (۷۶) خروجی این نمودار را نمایش می‌دهد. با توجه مشاهده می‌شود که K بهینه برابر با ده است و در این حالت دقت برابر با ۰.۶۴۴۴ خواهد شد.

```

● ● ●
1 # Results for plotting
2 mean_scores = grid.cv_results_['mean_test_score']
3 k_values = np.arange(1, 21)
4
5 # Plotting accuracy vs K
6 plt.figure(figsize=(10, 6))
7 plt.plot(k_values, mean_scores, marker='o')
8 plt.xlabel('Number of Neighbors: k')
9 plt.ylabel('Cross-Validated Accuracy')
10 plt.title('KNN Number of Neighbors vs. Accuracy')
11 plt.grid(True)
12 plt.xticks(k_values)
13 plt.show()
14
15 print("Best k:", best_k)
16 print("Best cross-validated accuracy:", best_score)

```

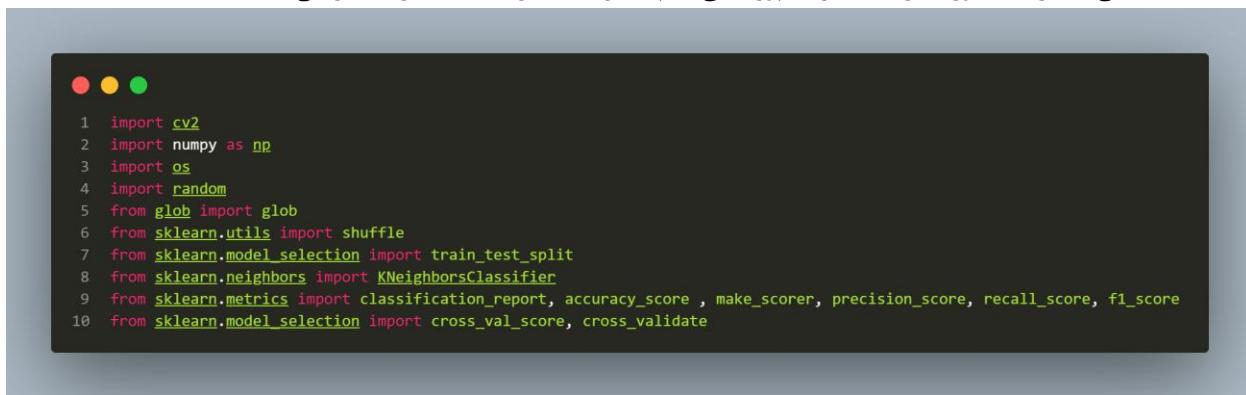
شکل ۷۴: کد رسم نمودار دقت بر اساس K



شکل ۷۵: نمودار دقت بر اساس K

۴-پاسخ سوال چهارم

ابتدا کتابخانه‌هایی که در ادامه مورد نیاز هستند را ایمپورت می‌کنیم. شکل (۷۶) این کتابخانه‌ها را نمایش می‌دهد.



```
1 import cv2
2 import numpy as np
3 import os
4 import random
5 from glob import glob
6 from sklearn.utils import shuffle
7 from sklearn.model_selection import train_test_split
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.metrics import classification_report, accuracy_score, make_scorer, precision_score, recall_score, f1_score
10 from sklearn.model_selection import cross_val_score, cross_validate
```

شکل (۷۶): وارد کردن کتابخانه‌های مورد نیاز

۴-۱-وارد کردن و پیش‌پردازش داده‌ها

با استفاده از یک تابع، دو عملکرد وارد کردن تصاویر و پیش‌پردازش توابع را انجام می‌دهیم. این تابع عکس‌ها را دریافت می‌کند و چند عملیات پیش‌پردازش که در ادامه به آنها پرداخته می‌شود را انجام می‌دهد و عکس آماده برای آموزش مدل را برمی‌گرداند. این رویکرد چندان همسو با کدنویسی تمیز و این قانون که هر تابع صرفاً یک عملیات را انجام دهد نیست ولی برای احرا در مقیاس کوچک موثر است.

حال به بررسی بخش به بخش این تابع می‌پردازیم.

- تابع `(*)` آرگومان `os.path.join(directory, f"{{label}}.*.jpg")` را دریافت می‌کند و در مسیر تخصیص داده شده به این آرگومان، دنبال یک الگوی خاص می‌گردد. بخش اول الگو با استفاده از `f string` از آرگومان `label` دریافت می‌شود. بخش دوم `*` است که یک `wild card` است و هر کاراکتری را قبول می‌کند و بخش سوم `.jpg` است که فرمت دریافتی را مشخص می‌کند. آرگومان `directory` مسیر فایل را دریافت می‌کند و آرگومان `label` به سطر هدف عکس اشاره دارد. این الگو در `glob_pattern` ذخیره می‌شود.

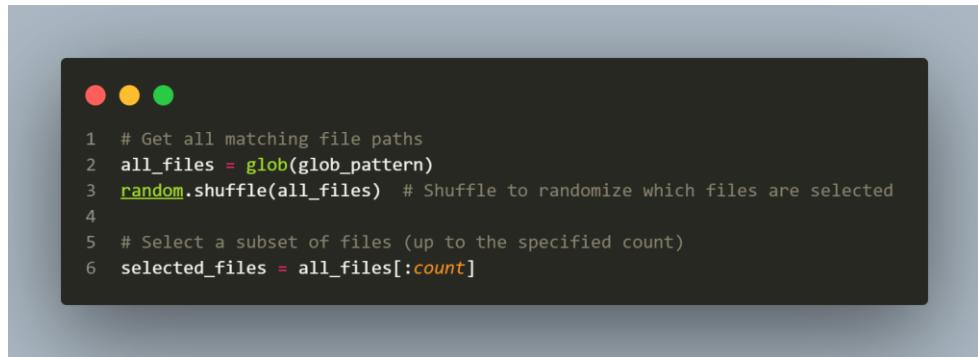
شکل (۷۷) این بخش از تابع را نمایش می‌دهد.



```
1 # Create a glob pattern to match all jpg images in the directory
2 glob_pattern = os.path.join(directory, f"{{label}}.*.jpg")
```

شکل (۷۷): تعریف الگوی دریافتی

- حال با استفاده از تابع `glob()` و پاس دادن الگوی ساخته شد، تمامی مسیرهایی که در پوشش مشخص شده هستند را دریافت می‌کنیم. سپس با استفاده از تابع `suffle()` این مسیرها را بر می‌زنیم تا انتخاب فایل‌ها به صورت تصادفی باشد. در ادامه به تعداد مشخص شده در آرگومان `count` مسیر عکس‌ها را انتخاب می‌کنیم. شکل (۷۸) نحوه انجام این کار را نمایش می‌دهد.
- سپس دو لیست خالی برای نگهداری جدایگانه متغیرها و سطر هدف آنها ایجاد می‌کنیم.



```

1 # Get all matching file paths
2 all_files = glob(glob_pattern)
3 random.shuffle(all_files) # Shuffle to randomize which files are selected
4
5 # Select a subset of files (up to the specified count)
6 selected_files = all_files[:count]

```

شکل ۷۸: خواندن و انتخاب تصادفی فایل‌ها

در قدم بعدی، در یک حلقه، به ازای هر مسیر دریافت شده، عملیات‌های زیر را انجام می‌دهیم:

۱-ابتدا با استفاده از تابع `imread` از این مسیر، عکس را به صورت رنگی می‌خوانیم و در متغیر `img` ذخیره می‌کنیم. آرگومان `file_path` مسیر را مشخص می‌کند و آرگومان `IMREAD_COLOR` `cv`.`cv2` را خواندن عکس را. در صورتی که عکس به درستی خوانده نشود، پیام خطایی نمایش داده می‌شود

۲-سپس عکس خوانده شده را با استفاده از تابع `cvtColor` و پاس دادن `img` به تابع و آرگومان `cv2.COLOR_BGR2GRAY` به صورت `gray` تبدیل می‌کنیم و نتیجه را در متغیر `img_gray` ذخیره می‌کنیم.

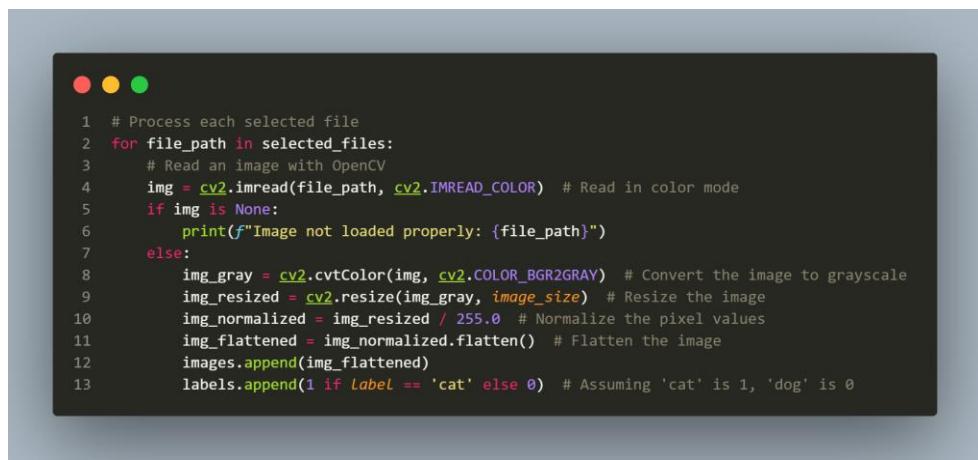
۳-سپس با استفاده از تابع `resize` عکس را مجدداً سایز بندی می‌کنیم. متغیر `image_size` که به این تابع پاس داده شده است، آرگومان دریافت شده توسط تابع اصلی است.

۴-عکس با ابعاد جدید را با تقسیم بر `255` نرمال می‌کنیم.

۵-سپس با استفاده از متد `flatten()`. ماتریس عکس را به یک بردار تک ستونی تبدیل می‌کنیم.

۶-در نهایت این بردار را در لیست ایجاد شده برای متغیرها ذخیره می‌کنیم. برای ذخیره لیبل، اگر لیبل مربوط به گربه بود، آن را با یک و اگر مربوط به سگ بود آن را با صفر در لیست مربوط به لیبل ذخیره می‌کنیم

شکل (۷۹) این حلقه را نمایش می‌دهد.



```

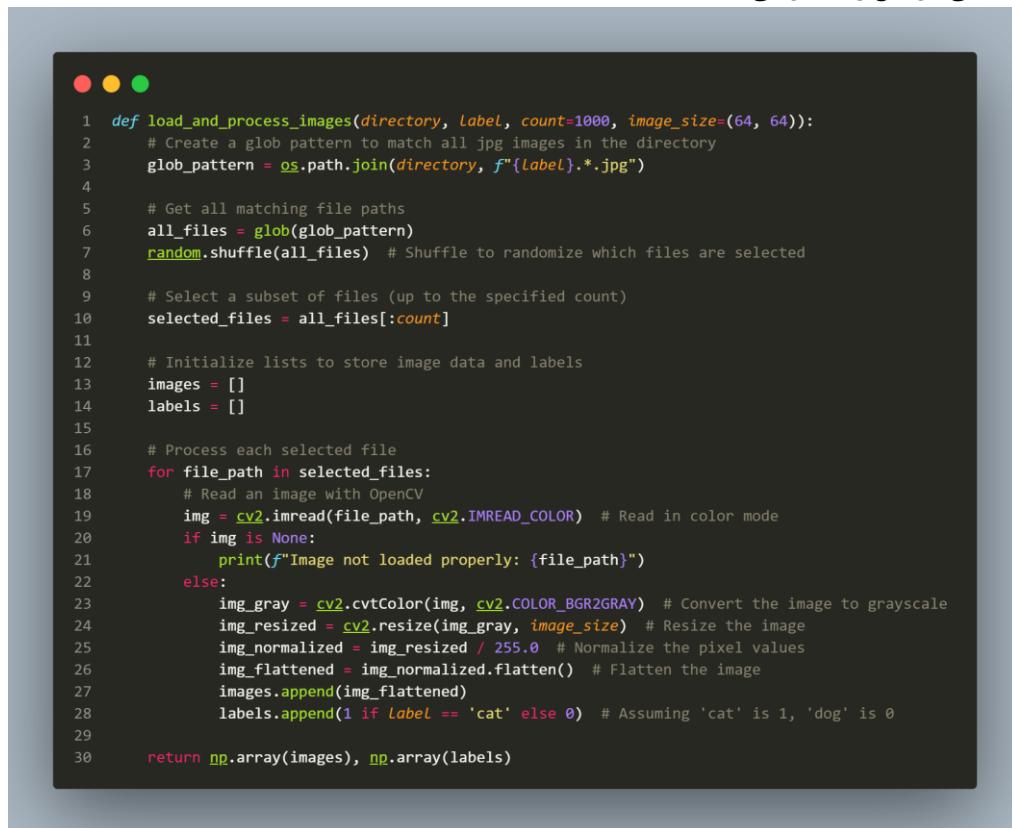
1 # Process each selected file
2 for file_path in selected_files:
3     # Read an image with OpenCV
4     img = cv2.imread(file_path, cv2.IMREAD_COLOR) # Read in color mode
5     if img is None:
6         print(f"Image not loaded properly: {file_path}")
7     else:
8         img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert the image to grayscale
9         img_resized = cv2.resize(img_gray, image_size) # Resize the image
10        img_normalized = img_resized / 255.0 # Normalize the pixel values
11        img_flattened = img_normalized.flatten() # Flatten the image
12        images.append(img_flattened)
13        labels.append(1 if label == 'cat' else 0) # Assuming 'cat' is 1, 'dog' is 0

```

شکل ۷۹: پیش‌برداش یک به یک عکس‌ها

در نهایت تابع لیست‌های ایجاد شده را به شکل آرایه‌های نامپایی بر می‌گرداند

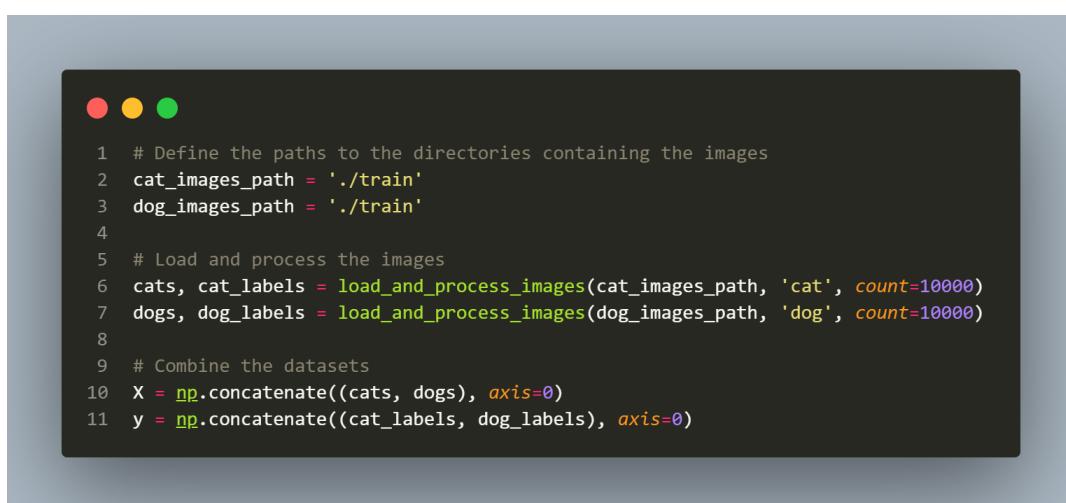
شکل (۸۰) نمای کلی این تابع را نمایش می‌دهد.



```
1 def load_and_process_images(directory, label, count=1000, image_size=(64, 64)):
2     # Create a glob pattern to match all jpg images in the directory
3     glob_pattern = os.path.join(directory, f"{label}.*.jpg")
4
5     # Get all matching file paths
6     all_files = glob(glob_pattern)
7     random.shuffle(all_files) # Shuffle to randomize which files are selected
8
9     # Select a subset of files (up to the specified count)
10    selected_files = all_files[:count]
11
12    # Initialize lists to store image data and labels
13    images = []
14    labels = []
15
16    # Process each selected file
17    for file_path in selected_files:
18        # Read an image with OpenCV
19        img = cv2.imread(file_path, cv2.IMREAD_COLOR) # Read in color mode
20        if img is None:
21            print(f"Image not loaded properly: {file_path}")
22        else:
23            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert the image to grayscale
24            img_resized = cv2.resize(img_gray, image_size) # Resize the image
25            img_normalized = img_resized / 255.0 # Normalize the pixel values
26            img_flattened = img_normalized.flatten() # Flatten the image
27            images.append(img_flattened)
28            labels.append(1 if label == 'cat' else 0) # Assuming 'cat' is 1, 'dog' is 0
29
30    return np.array(images), np.array(labels)
```

شکل ۸۰: تابع خواندن و پیش‌پردازش عکس‌ها

در قدم بعد، ابتدا آدرس هر یک از دیتاست‌های موجود را در یک متغیر ذخیره می‌کنیم. فایل نوتبوک به عنوان root در نظر گرفته می‌شود، در نتیجه اگر فولد دیتاست در کنار فایل نوتبوک باشد، آدرس دهی به شکل './train' انجام می‌شود. سپس با استفاده از تابع ایجاد شده، ده هزار عکس گربه، و ده هزار عکس سگ را می‌خوانیم و در لیست‌های متناظر با متغیرها و سطر هدف هر یک ذخیره می‌کنیم. سپس لیست‌ها را دو تا با هم ادغام می‌کنیم تا دیتاست نهایی ایجاد شود. شکل (۸۱) نحوه انجام این کار را نمایش می‌دهد.



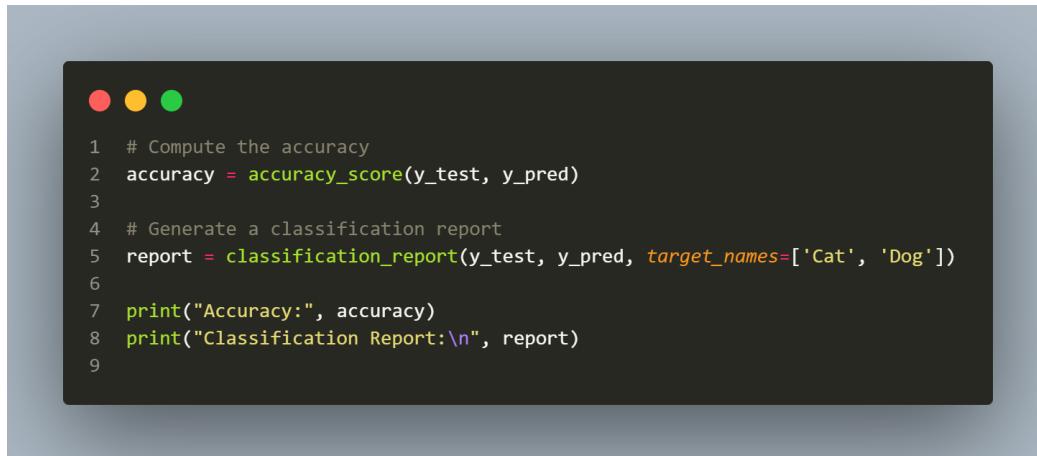
```
1 # Define the paths to the directories containing the images
2 cat_images_path = './train'
3 dog_images_path = './train'
4
5 # Load and process the images
6 cats, cat_labels = load_and_process_images(cat_images_path, 'cat', count=10000)
7 dogs, dog_labels = load_and_process_images(dog_images_path, 'dog', count=10000)
8
9 # Combine the datasets
10 X = np.concatenate((cats, dogs), axis=0)
11 y = np.concatenate((cat_labels, dog_labels), axis=0)
```

شکل ۸۱: فراخوانی و پیش‌پردازش عکس‌ها و ایجاد دیتاست نهایی

حال آمده آموزش مدل بر روی عکس‌ها هستیم.

KNN-۲-۴ آموزش

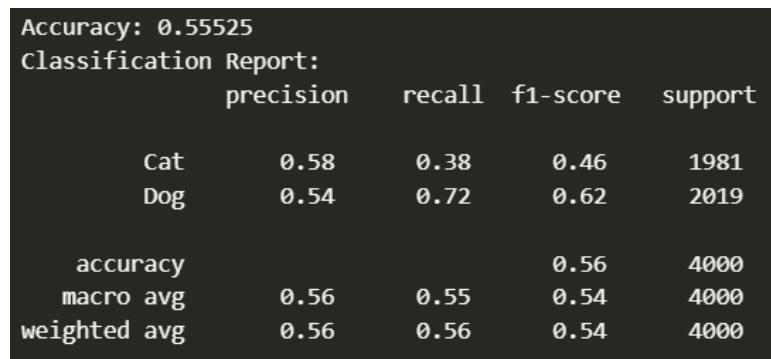
ابتدا دیتاست را به دو دسته آموزشی و آزمایشی تقسیم‌بندی می‌کنیم. هشتاد درصد داده‌ها را برای آموزش و بیست درصد را برای آزمایش کنار می‌گذاریم. سپس مدل را با پارامتر پنج همسایه آموزش می‌دهیم و روی داده‌های آزمایشی تست می‌کنیم. در ادامه با استفاده ازتابع classification_report() و پاس دادن پیش‌بینی‌ها و لیبل‌های سطر هدف و نام کلاس‌ها، یک گزارش از معیارهای عملکرد مدل درست می‌کنیم. شکل (۸۲) کد مربوط به این بخش را نمایش می‌دهد.



```
1 # Compute the accuracy
2 accuracy = accuracy_score(y_test, y_pred)
3
4 # Generate a classification report
5 report = classification_report(y_test, y_pred, target_names=['Cat', 'Dog'])
6
7 print("Accuracy:", accuracy)
8 print("Classification Report:\n", report)
9
```

شکل (۸۲): کد نمایش گزارش عملکرد

همچنین شکل (۸۳) گزارش درست شده را نمایش می‌دهد.



Classification Report:				
	precision	recall	f1-score	support
Cat	0.58	0.38	0.46	1981
Dog	0.54	0.72	0.62	2019
accuracy			0.56	4000
macro avg	0.56	0.55	0.54	4000
weighted avg	0.56	0.56	0.54	4000

شکل (۸۳): گزارش عملکرد مدل KNN

Cross Validation از ۳-۴ استفاده

مجددآ مدل را با همان پارامتر قبلی ایجاد می‌کنیم. برای محاسبه معیارهای گفته شده به همراه انجراف استانداردشان، ابتدا یک دیکشنری از معیارها ایجاد می‌کنیم که کلیدها نام هر معیار و value‌ها یک تابع make_scoring() هستند که آرگومان هر کدام متناظر با معیار تعریف شده در کلید دیکشنری است. شکل (۸۴) این دیکشنری را نمایش می‌دهد.



```
● ● ●  
1 scoring = {  
2     'accuracy': 'accuracy',  
3     'precision': make_scorer(precision_score, average='macro'),  
4     'recall': make_scorer(recall_score, average='macro'),  
5     'f1': make_scorer(f1_score, average='macro')  
6 }
```

شکل ۸۴: دیکشنری محاسبه معیارهای ارزیابی

سپس با استفاده از تابع `cross_validate` و پاس دادن مدل، دیتابست، تعداد بخش‌ها که برابر ده است و دیکشنری معیارها، Cross Validation را انجام می‌دهیم.



```
● ● ●  
1 results = cross_validate(knn, X, y, cv=10, scoring=scoring)
```

شکل ۸۵: انجام Cross Validation

سپس نتایج به دست آمده را نمایش می‌دهیم. شکل (۸۶) این نتایج را نشان می‌دهد.

```
Average Accuracy: 0.56 (+/- 0.02)  
Average Precision: 0.57 (+/- 0.02)  
Average Recall: 0.56 (+/- 0.02)  
Average F1-score: 0.54 (+/- 0.02)
```

شکل ۸۶: نتایج به دست آمده از Cross Validation

مشاهده می‌شود که نتایج به دست آمده بسیار ضعیف هستند. در واقع عملکرد KNN در دسته بندی عکس‌ها تنها اندکی بهتر از دسته بندی عکس‌ها با استفاده از پرتاب یک سکه است.