

تمرین دوم هوش مصنوعی، رگرسیون

تمرین دوم

آرین محمدخانی

Arian MohammadKhani

چکیده و نکات

برای این تمرین، فایل کد نویسی، فایل Jupiter notebook پی دی اف و فایل گزارش ارائه داده شده است. همچنین این تمرین در سایت GitHub جهت امتیاز بیشتر آپلود شده است. تمامی خروجی های مورد نیاز در این گزارش ارائه شده است اما جهت نمایش بهتر خروجی ها بهتر است به گزارش Jupiter مراجعه شود.

- برای نمایش بهتر خروجی ها، فایل (HW2-jupyter-report) مشاهده شود.
- جهت بررسی کد پایتون، فایل (hw2.py)
- جهت خواندن گزارش، فایل (hw2-report)
- <https://github.com/ArianAZH/AI-exercise-regression>

۱. مقدمه

هدف این تمرین، برآوردهای هزینه‌ی خانه‌ها با توجه به معیارهای مختلف است. برای تخمین این هزینه، نیاز به آشنایی با روش‌های پیش‌پردازش داده‌ها و همچنین ارزیابی کارآیی مدل‌های مختلف رگرسیون وجود دارد. داده‌هایی که برای این کار در نظر گرفته شده‌اند شامل ۲۹۳۰ نمونه هستند که در آن‌ها مقادیر پرت (Outlier) و مقادیر ناموجود (Missing Values) نیز وجود دارد. بنابراین، پیش از آموزش مدل، باید با استفاده از روش‌های مناسب پیش‌پردازش، داده‌ها اصلاح شوند.

در زبان پایتون، برای استفاده از توابع کتابخانه‌های مختلف، از دستور `import (function) from (library)` استفاده می‌شود. در برخی موارد، از علامت (*) نیز استفاده شده است که به معنای وارد کردن تمام توابع موجود در آن کتابخانه است. در تمامی بخش‌های این تمرین، کدهای مورد استفاده، فرضیات و خروجی‌ها به صورت کامل نمایش داده و تحلیل شده‌اند.

```
from pandas import *
from numpy import *
from seaborn import heatmap
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import PolynomialFeatures, LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
```

۲. بخش یک

در ابتدا داده‌ها از فایل `Housing.csv` با استفاده از تابع `pandas.read_csv` بارگذاری شده‌اند و در متغیری به نام `data_frame` ذخیره شده‌اند. این فایل شامل اطلاعات مربوط به ویژگی‌های مختلف خانه‌هاست که در ادامه مورد بررسی قرار خواهند گرفت. سپس با استفاده از متدهای `info()`، اطلاعات کلی از ساختار دیتافریم از جمله تعداد ردیف‌ها و ستون‌ها، نوع داده‌ای هر ستون و تعداد مقادیر غیرتھی به دست آمده است. این اطلاعات برای آشنایی اولیه با داده‌ها و شناسایی ستون‌هایی که ممکن است دارای مقادیر گمشده باشند، بسیار مفید است. در پایان، با اجرای تابع `head()` پنج ردیف اول دیتافریم نمایش داده شده است تا نمایی کلی از شکل و محتوای

داده‌ها به دست آید. این مرحله به عنوان بررسی اولیه داده‌ها (Data Inspection) شناخته می‌شود و نقش پایه‌ای در آماده‌سازی داده‌ها برای تحلیل‌های بعدی ایفا می‌کند.

```
data_frame = read_csv('Housing.csv')
data_frame.info()
print(data_frame.head())
```

و بخشی از خروجی‌ها به شکل زیر می‌باشد. (خروجی‌های کامل در فایل گزارش جوپیتر می‌باشد.)

<class 'pandas.core.frame.DataFrame'>																
RangeIndex: 2930 entries, 0 to 2929																
Data columns (total 82 columns):																
#	Column	Non-Null Count	Dtype													
0	Order	2930	non-null		int64											
1	PID	2930	non-null		int64											
2	MS SubClass	2930	non-null		int64											
3	MS Zoning	2930	non-null		object											
4	Lot Frontage	2440	non-null		float64											
5	Lot Area	2930	non-null		int64											
6	Street	2930	non-null		object											
7	Alley	198	non-null		object											
8	Lot Shape	2930	non-null		object											
9	Land Contour	2930	non-null		object											
10	Utilities	2930	non-null		object											
11	Lot Config	2930	non-null		object											
12	Land Slope	2930	non-null		object											
13	Neighborhood	2930	non-null		object											
14	Condition 1	2930	non-null		object											
15	Condition 2	2930	non-null		object											
16	Bldg Type	2930	non-null		object											
17	House Style	2930	non-null		object											

Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010	WD	Normal
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal
2	3	526351010	20	RL	81.0	14287	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal

خروجی ۱

۳. بخش دوم

در مرحله‌ی پیش‌پردازش داده‌ها، ابتدا داده‌های پرت شناسایی و حذف می‌شوند و سپس مقادیر گمشده مورد بررسی قرار می‌گیرند. ترتیب انجام این دو مرحله از اهمیت بالایی برخوردار است، چراکه وجود داده‌های پرت می‌تواند شاخص‌های آماری مانند میانه و چارک‌ها را که در تشخیص مقادیر گمشده یا انجام جایگزینی مؤثر هستند، به‌طور نادرست تحت تأثیر قرار دهد.

در ابتدا، برای آن که داده‌های متنی (objective) قابل پردازش توسط مدل‌های یادگیری ماشین باشند، با استفاده از کلاس LabelEncoder از کتابخانه‌ی sklearn.preprocessing، مقادیر متنی به مقادیر عددی تبدیل شدند. این کار بر روی تمامی ستون‌هایی که نوع داده‌ای آن‌ها object بوده، اعمال شده است.

```
label_encoder = LabelEncoder()  
for column in data_frame.select_dtypes(include='object').columns:  
    data_frame[column] = label_encoder.fit_transform(data_frame[column])
```

سپس مقادیر گمشده با استفاده از متدهای isnull() و sum() بررسی شدند و تعداد مقادیر گمشده در هر ستون محاسبه و چاپ گردید.

```
missing_v = data_frame.isnull().sum()  
missing_v = missing_v[missing_v > 0]  
print(missing_v)
```

LotFrontage	490
MasVnrArea	23
BsmtFinSF_1	1
BsmtFinSF_2	1
BsmtUnfSF	1
TotalBsmtSF	1
BsmtFullBath	2
BsmtHalfBath	2
GarageYrBlt	159
GarageCars	1
GarageArea	1
dtype:	int64

خروجی ۲

برای شناسایی داده‌های پرت، از روش IQR (Interquartile Range) استفاده شد. این روش بر اساس فاصله‌ی بین چارک اول (Q1) و چارک سوم (Q3)، بازه‌ای قابل قبول برای داده‌ها تعیین می‌کند. مقادیری که خارج از این بازه قرار دارند به عنوان داده‌ی پرت شناسایی می‌شوند. این فرایند در قالب تابعی به نام detect_outliers_iqr

پیاده‌سازی شده تا کد ساختاریافته‌تر، قابل استفاده مجدد و خواناتر باشد. نتیجه‌ی این تابع شامل تعداد و درصد داده‌های پرت در هر ستون عددی است که چاپ می‌شود.

```
def detect_outliers_iqr(df):
    outlier_info = {}
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    for col in numeric_columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
        if not outliers.empty:
            outlier_info[col] = {
                'count': len(outliers),
                'percent': round(len(outliers) / len(df) * 100, 2)
            }
    return outlier_info

outliers_summary = detect_outliers_iqr(data_frame)
for col, info in outliers_summary.items():
    print(f"🔍 Column: {col} | Outliers: {info['count']} rows ({info['percent']}%)")
```

```
🔍 Column: MS SubClass | Outliers: 208 rows (7.1%)
🔍 Column: Lot Frontage | Outliers: 187 rows (6.38%)
🔍 Column: Lot Area | Outliers: 127 rows (4.33%)
🔍 Column: Overall Qual | Outliers: 4 rows (0.14%)
🔍 Column: Overall Cond | Outliers: 252 rows (8.6%)
🔍 Column: Year Built | Outliers: 9 rows (0.31%)
🔍 Column: Mas Vnr Area | Outliers: 200 rows (6.83%)
🔍 Column: BsmtFin SF 1 | Outliers: 15 rows (0.51%)
🔍 Column: BsmtFin SF 2 | Outliers: 351 rows (11.98%)
🔍 Column: Bsmt Unf SF | Outliers: 56 rows (1.91%)
🔍 Column: Total Bsmt SF | Outliers: 123 rows (4.2%)
🔍 Column: 1st Flr SF | Outliers: 43 rows (1.47%)
🔍 Column: 2nd Flr SF | Outliers: 8 rows (0.27%)
🔍 Column: Low Qual Fin SF | Outliers: 48 rows (1.37%)
🔍 Column: Gr Liv Area | Outliers: 75 rows (2.56%)
🔍 Column: Bsmt Full Bath | Outliers: 2 rows (0.07%)
🔍 Column: Bsmt Half Bath | Outliers: 175 rows (5.97%)
🔍 Column: Full Bath | Outliers: 4 rows (0.14%)
🔍 Column: Bedroom AbvGr | Outliers: 78 rows (2.66%)
🔍 Column: Kitchen AbvGr | Outliers: 134 rows (4.57%)
🔍 Column: TotRms AbvGrd | Outliers: 51 rows (1.74%)
🔍 Column: Fireplaces | Outliers: 13 rows (0.44%)
🔍 Column: Garage Yr Blt | Outliers: 3 rows (0.1%)
🔍 Column: Garage Cars | Outliers: 17 rows (0.56%)
🔍 Column: Garage Area | Outliers: 42 rows (1.43%)
🔍 Column: Wood Deck SF | Outliers: 67 rows (2.29%)
🔍 Column: Open Porch SF | Outliers: 159 rows (5.43%)
🔍 Column: Enclosed Porch | Outliers: 459 rows (15.67%)
🔍 Column: 3Ssn Porch | Outliers: 37 rows (1.26%)
🔍 Column: Screen Porch | Outliers: 256 rows (8.74%)
🔍 Column: Pool Area | Outliers: 13 rows (0.44%)
🔍 Column: Misc Val | Outliers: 103 rows (3.52%)
🔍 Column: SalePrice | Outliers: 137 rows (4.68%)
```

خروجی ۳

در گام بعدی، برای حذف داده‌های پرت از دیتافریم، تابعی به نام `remove_outliers_iqr` طراحی شده است. این تابع مشابه تابع شناسایی، بر اساس روش IQR عمل می‌کند، با این تفاوت که نمونه‌های پرت را به‌طور کامل از دیتافریم حذف می‌نماید. خروجی این تابع دیتافریمی پاک‌سازی شده از داده‌های پرت است که در متغیر `modified_df` ذخیره شده و برای مراحل بعدی تحلیل مورد استفاده قرار می‌گیرد.

استفاده از توابع مجزا برای شناسایی و حذف داده‌های پرت باعث افزایش modularity (ماژولار بودن)، امکان استفاده مجدد، و سادگی در تست و اشکال‌زدایی کد می‌شود. به علاوه، این رویکرد منجر به افزایش خوانایی و حرفاء‌تر شدن ساختار پروژه می‌گردد.

```
def remove_outliers_iqr(df):
    df_clean = df.copy()
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

    for col in numeric_columns:
        Q1 = df_clean[col].quantile(0.25)
        Q3 = df_clean[col].quantile(0.75)
        iqr = Q3 - Q1
        lower_bound = Q1 - 1.5 * iqr
        upper_bound = Q3 + 1.5 * iqr
        df_clean = df_clean[(df_clean[col] >= lower_bound) & (df_clean[col] <= upper_bound)]
    return df_clean

data_frame_cleaned = remove_outliers_iqr(data_frame)
modified_df = data_frame_cleaned
```

در این بخش از پیش‌پردازش، تمرکز بر مدیریت دقیق مقادیر گمشده و حذف ویژگی‌های کمارزش بوده است. ابتدا با پیمایش تمام ستون‌های دیتافریم، تعداد مقادیر گمشده در هر ستون بررسی شد. در صورتی که مقدار گمشده در یک ستون بیش از ۱۰۰۰ مقدار بود، آن ستون به طور کامل از دیتافریم حذف گردید، چراکه حجم بالای داده‌های ناقص در یک ویژگی می‌تواند تحلیل‌ها را دچار خطا کند و ارزش نگهداری آن ویژگی را زیر سوال ببرد. اما اگر تعداد مقادیر گمشده کمتر از این آستانه بود، بسته به نوع داده‌ی ستون، مقدار مناسبی برای جایگزینی انتخاب شد. برای ستون‌های عددی (ستون‌هایی با تایپ `float64` و `int64`، میانه‌ی داده‌ها مورد استفاده قرار گرفت تا اثر داده‌های پرت کاهش یابد، در حالی که برای ستون‌های غیرعددی یا طبقه‌ای، از مقدار پر تکرار (مد) به عنوان جایگزین استفاده شد.

علاوه بر این، تنوع اطلاعاتی در هر ستون نیز مورد بررسی قرار گرفت. در صورتی که بیش از ۷۰ درصد مقادیر یک ستون یکسان بودند، آن ستون نیز حذف شد. این تصمیم با هدف جلوگیری از ورود ویژگی‌هایی به مدل اتخاذ شده که اگرچه ممکن است از نظر آماری معتبر باشند، اما به دلیل نبود تنوع کافی، نمی‌توانند اطلاعات معناداری برای یادگیری مدل فراهم کنند. در ادامه، ستون "Order" نیز به صورت صریح حذف گردید، چراکه این ستون تنها نقش شماره‌گذاری یا مرتب‌سازی داشته و هیچ ارزش تحلیلی برای مدل‌سازی نداشته است و در پایان این مراحل، یک بررسی مجدد برای اطمینان از عدم وجود مقادیر گمشده‌ی باقی‌مانده انجام گرفت.

```
for column in modified_df.columns:  
    missing_value = modified_df[column].isnull().sum()  
    if missing_value > 0:  
        if missing_value > 1000:  
            modified_df.drop(columns=[column], inplace=True)  
        else:  
            if modified_df[column].dtype == 'float64' and modified_df[column].dtype == 'int64':  
                median_data = modified_df[column].median()  
                modified_df[column].fillna(median_data, inplace=True)  
            else:  
                mode_data = modified_df[column].mode()[0]  
                modified_df[column].fillna(mode_data, inplace=True)  
    top_freq_ratio = modified_df[column].value_counts(normalize=True).values[0]  
    if top_freq_ratio > 0.70:  
        modified_df.drop(columns=[column], inplace=True)  
  
modified_df.drop(columns=["Order"], inplace=True)  
missing_v = modified_df.isnull().sum()  
missing_v = missing_v[missing_v > 0]  
print(missing_v)
```

Series([], dtype: int64)

خروجی :

۴. بخش سوم

اجرای دستور `print(modified_df.describe())` خلاصه‌ای آماری از تمام ستون‌های عددی دیتافریم `modified_df` را نمایش می‌دهد. این خلاصه شامل اطلاعات مهمی مثل میانگین، انحراف معیار، حداقل، چارک‌ها (٪.۲۵، ٪.۵۰، ٪.۷۵) و حداقل مقدار برای هر ویژگی عددی است.

```
print(modified_df.describe())
```

	PID	MS SubClass	Lot Frontage	Lot Area	Lot Shape	Neighborhood	House Style	Overall Qual	Year Built	Year Remod/Add	...	Fireplace Qu	Ga
count	8.620000e+02	862.000000	862.000000	862.000000	862.000000	862.000000	862.000000	862.000000	862.000000	862.000000	...	862.000000	862
mean	6.996168e+08	49.071926	68.185615	9067.616009	1.969838	13.663573	3.024362	6.344548	1983.895502	1989.667053	...	3.841087	1986
std	1.880622e+08	31.700866	15.062406	2493.452117	1.414713	7.581722	1.815051	1.269068	25.836810	20.248269	...	1.389870	23
min	5.263021e+08	20.000000	30.000000	3010.000000	0.000000	0.000000	0.000000	2.000000	1890.000000	1950.000000	...	0.000000	1895
25%	5.283315e+08	20.000000	60.000000	7608.250000	0.000000	6.000000	2.000000	5.000000	1963.000000	1973.000000	...	2.000000	1988
50%	5.353262e+08	50.000000	70.000000	9030.500000	3.000000	15.000000	2.000000	6.000000	1998.000000	2000.000000	...	5.000000	1998
75%	9.071870e+08	60.000000	78.000000	10572.000000	3.000000	20.000000	5.000000	7.000000	2005.000000	2006.000000	...	5.000000	2005
max	9.241510e+08	120.000000	110.000000	16285.000000	3.000000	27.000000	7.000000	10.000000	2010.000000	2010.000000	...	5.000000	2010

خروجی ۵

۵. بخش چهارم

در این بخش از کد، ابتدا یک ماتریس همبستگی با استفاده از متدها (`corr()`). محاسبه می‌شود که روابط بین تمامی ویژگی‌های عددی موجود در دیتافریم `modified_df` را نشان می‌دهد. این ماتریس شامل ضریب همبستگی پیرسون است که برای هر جفت از ویژگی‌ها، درجه‌ای از ارتباط خطی بین آن‌ها را مشخص می‌کند. سپس، برای تمرکز روی ارتباطات بین ویژگی‌ها و قیمت فروش (`SalePrice`)، یک ستون از ماتریس همبستگی استخراج می‌شود. این ستون، همبستگی هر ویژگی با `SalePrice` را نشان می‌دهد. در مرحله بعد، با استفاده از متدهای `sort_values(ascending=False)` روابط همبستگی به ترتیب نزولی مرتب می‌شوند. این کار به ما کمک می‌کند تا روابط قوی‌تر را سریع‌تر شناسایی کنیم. در نهایت، از آنجا که ممکن است برخی از روابط همبستگی ضعیف یا غیرقابل توجه باشند، فقط روابطی که دارای همبستگی بالاتر از ۰.۵ (چه مثبت و چه منفی) هستند فیلتر می‌شوند.

```
correlation_matrix = modified_df.corr(numeric_only=True)

correlation_with_price = correlation_matrix['SalePrice']

sorted_corr = correlation_with_price.sort_values(ascending=False)

strong_corr = sorted_corr[abs(sorted_corr) > 0.5]
```

SalePrice	1.000000
Overall Qual	0.831094
Gr Liv Area	0.724245
Garage Cars	0.717507
Year Built	0.705330
Garage Area	0.703422
Garage Yr Blt	0.669893
Full Bath	0.665367
Year Remod/Add	0.639692
Foundation	0.610429
Total Bsmt SF	0.580778
1st Flr SF	0.547298
TotRms AbvGrd	0.534539
Garage Finish	-0.529050
Heating QC	-0.530653
Kitchen Qual	-0.659560
Bsmt Qual	-0.668972
Exter Qual	-0.709607
Name: SalePrice, dtype: float64	

۶. خروجی

۶. بخش پنجم

در ابتدا، اندازه‌ی نمودار با استفاده از دستور plt.figure(figsize=(20,15)) تنظیم می‌شود تا نمودار بزرگ‌تر و واضح‌تر نمایش داده شود. این کار بهویژه زمانی مفید است که دیتافریم شامل تعداد زیادی ویژگی است و برای مشاهده‌ی بهتر ارتباطات بین آن‌ها نیاز به نمایی بزرگ‌تر داریم. پس از تنظیم اندازه، ماتریس همبستگی بهصورت heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5) رسم می‌شود. در اینجا، ماتریس همبستگی که پیش‌تر محاسبه شده، بهعنوان ورودی به تابع داده می‌شود. پارامتر annot=True باعث می‌شود که مقادیر همبستگی در هر خانه‌ی ماتریس نمایش داده شوند، بنابراین می‌توانیم دقیقاً میزان ارتباط بین هر جفت ویژگی را مشاهده کنیم. رنگبندی برای تفکیک همبستگی‌ها با استفاده از رنگ‌های سرد (برای همبستگی منفی) و گرم (برای همبستگی مثبت) استفاده می‌شود، که تشخیص درجه‌ی ارتباطات را آسان‌تر می‌کند. همچنین، با استفاده از پارامتر linewidths=0.5 فاصله‌ی بین خانه‌های ماتریس تنظیم می‌شود تا این خانه‌ها از یکدیگر تفکیک شوند و

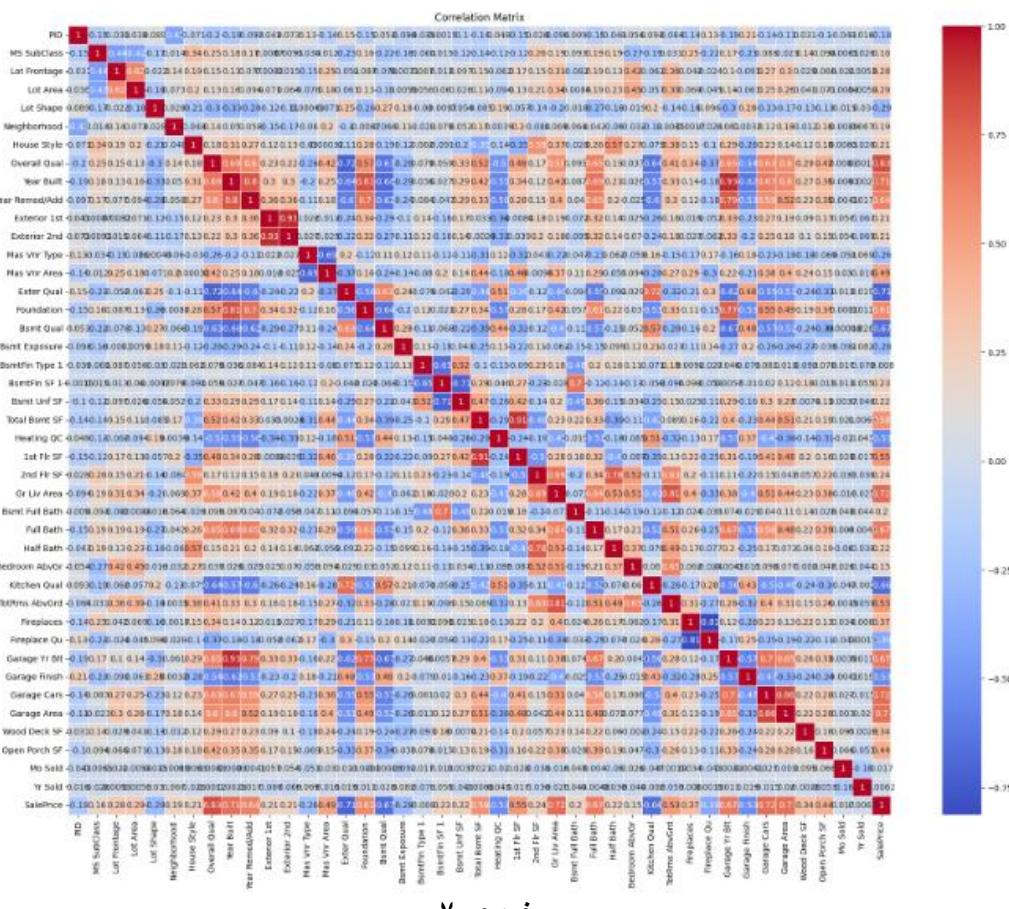
نمایش واضح‌تری داشته باشیم. در نهایت، با اضافه کردن عنوان Correlation Matrix به نمودار و نمایش آن با استفاده از plt.show() نمودار نهایی به‌طور کامل نمایش داده می‌شود.

```
plt.figure(figsize=(20,15))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()
```

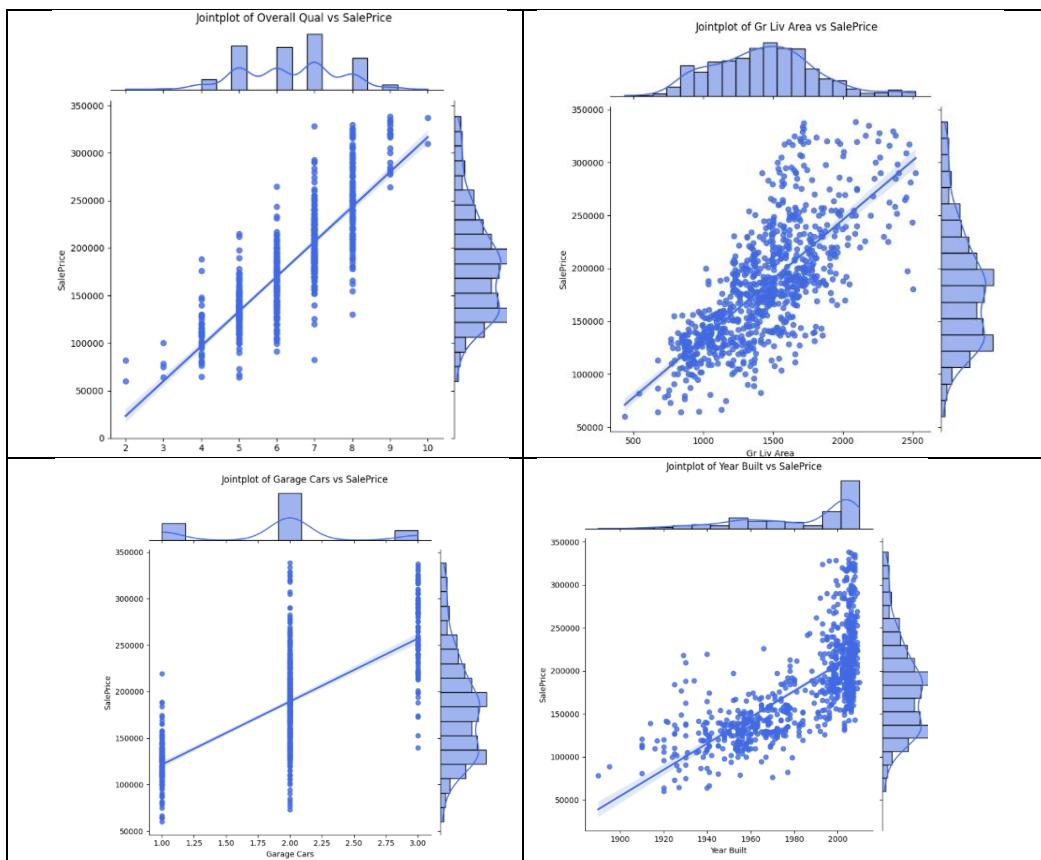


خروجی ۷

همچنین طبق خواسته سوال، نمودار jointplot معیارهای مختلف نسبت به هزینه با کد زیر اجرا می شود. (در اینجا تنها چهار نمونه نشان داده شده است.)

```
important_features = strong_corr.drop('SalePrice').index
top_features = important_features[:3]

for feature in top_features:
    plot = sns.jointplot(data=modified_df, x=feature, y='SalePrice', kind='reg', height=7,
                          color='royalblue')
    plot.fig.suptitle(f'Jointplot of {feature} vs SalePrice', y=1.02)
    plt.show()
```



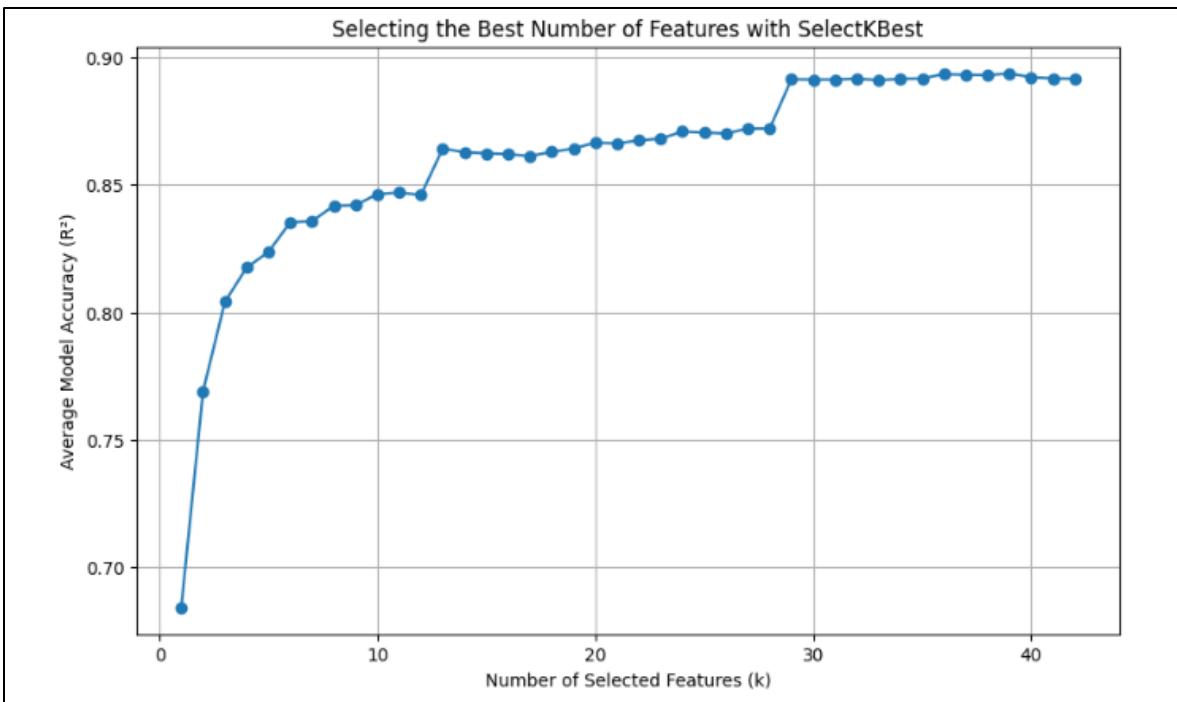
خروجی ۸

۷. بخش ششم

در این بخش از کد، هدف انتخاب بهترین تعداد ویژگی‌ها برای مدل رگرسیون خطی با استفاده از روش SelectKBest است که به کمک آن می‌توانیم ویژگی‌های مؤثرتر را برای پیش‌بینی قیمت فروش خانه شناسایی کنیم. ابتدا، داده‌های ورودی X با حذف ستون هدف (SalePrice) و هدف y که همان قیمت فروش است، آماده می‌شود. سپس با استفاده ازتابع get_dummies(X, drop_first=True)، متغیرهای طبقه‌ای به متغیرهای عددی تبدیل می‌شوند تا مدل بتواند به طور صحیح از آن‌ها استفاده کند. در ادامه، یک حلقه برای آزمایش تعداد مختلفی از ویژگی‌ها انجام می‌شود. برای هر مقدار k از تعداد ویژگی‌ها (از ۱ تا ۴۳)، ابتدا از SelectKBest برای انتخاب بهترین k ویژگی با استفاده از آزمون f_regression (که برای ویژگی‌های عددی است) استفاده می‌شود. سپس مدل رگرسیون خطی (LinearRegression) بر روی ویژگی‌های انتخابی اجرا می‌شود و با استفاده از ارزیابی اعتبار متقابل(cross_val_score)، میانگین دقت مدل با استفاده از معیار R^2 محاسبه می‌شود. این دقت در یک لیست scores ذخیره می‌شود. در نهایت، نتایج در یک نمودار رسم می‌شود که در آن محور افقی نشان‌دهنده تعداد ویژگی‌های انتخابی (k) و محور عمودی دقت مدل (R^2) است. این نمودار به ما کمک می‌کند تا بهترین تعداد ویژگی‌ها را برای مدل خود شناسایی کنیم. در این نمودار، افزایش یا کاهش دقت مدل در اثر تغییر تعداد ویژگی‌ها به وضوح قابل مشاهده است، و می‌توانیم تعداد بهینه ویژگی‌ها را برای داشتن بهترین دقت مدل انتخاب کنیم.

```
X = modified_df.drop('SalePrice', axis=1)
y = modified_df['SalePrice']
X = get_dummies(X, drop_first=True)
scores = []
k_values = range(1, min(len(X.columns),44))
for k in k_values:
    selector = SelectKBest(score_func=f_regression, k=k)
    X_selected = selector.fit_transform(X, y)
    model = LinearRegression()
    cv_score = cross_val_score(model, X_selected, y, cv=5, scoring='r2').mean()
    scores.append(cv_score)

plt.figure(figsize=(10, 6))
plt.plot(k_values, scores, marker='o')
plt.xlabel('Number of Selected Features (k)')
plt.ylabel('Average Model Accuracy ( $R^2$ )')
plt.title('Selecting the Best Number of Features with SelectKBest')
plt.grid(True)
plt.show()
```



خروجی ۹

۸. بخش هفتم

در این بخش از کد، داده‌ها به دو مجموعه‌ی آموزشی و تست تقسیم می‌شوند تا فرآیند آموزش و ارزیابی مدل به‌طور مؤثری انجام شود. با استفاده از دستور `train_test_split(X, y, test_size=0.25, random_state=42)`، داده‌ها به‌گونه‌ای تقسیم می‌شوند که ۷۵٪ از داده‌ها برای آموزش مدل و ۲۵٪ باقی‌مانده برای ارزیابی عملکرد مدل استفاده می‌شود. این تقسیم‌بندی با توجه به خواسته مسئله انجام می‌شود. این تقسیم‌بندی به مدل این امکان را می‌دهد که بر روی داده‌های آموزشی آموزش بییند و سپس با استفاده از داده‌های تست که از پیش مشاهده نکرده است، ارزیابی شود. این روش کمک می‌کند تا از مشکلاتی مانند overfitting جلوگیری شود و دقت مدل در پیش‌بینی‌های آینده به‌طور واقعی ارزیابی گردد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

۹. بخش هشتم

در این بخش از کد، چهار مدل مختلف رگرسیون برای پیش‌بینی قیمت فروش خانه بر اساس ویژگی‌های ورودی آموزش داده می‌شوند: رگرسیون خطی (Linear Regression)، لاسو (Lasso Regression)، ریج (Ridge Regression) و رگرسیون چندجمله‌ای (Polynomial Regression). هر کدام از این مدل‌ها ویژگی‌های خاص خود را دارند که می‌توانند در شرایط مختلف به بهبود پیش‌بینی‌ها کمک کنند.

رگرسیون خطی: این مدل یک مدل ساده و پایه است که در آن رابطه‌ای خطی بین ویژگی‌های ورودی (X_train) و هدف (y_train) برقرار می‌شود. با استفاده از دستور linear_model.fit(X_train, y_train) مدل رگرسیون خطی بر اساس داده‌های آموزشی آموزش داده می‌شود.

رگرسیون لاسو: لاسو نوعی از رگرسیون خطی است که در آن از تکنیک کاهش جرم (regularization) استفاده می‌شود تا ویژگی‌های غیرمهم حذف شوند. این امر باعث جلوگیری از overfitting و بهبود تعمیم‌پذیری مدل می‌شود. در اینجا، پارامتر alpha=1.0 شدت کاهش جرم را تنظیم می‌کند و مدل با آموزش داده می‌شود. پارامتر آلفا با توجه به مثال‌های موجود در سایت lasso_model.fit(X_train, y_train) راهنمای انتهاب شده است.

رگرسیون ریج: مانند لاسو، ریج هم از کاهش جرم استفاده می‌کند، ولی تفاوت اصلی این است که ریج از مجموع مربعات ضرایب مدل برای کاهش جرم استفاده می‌کند و نه از مقادیر مطلق ضرایب. این مدل به ویژه زمانی مفید است که ویژگی‌های زیادی در مدل وجود دارند که ممکن است همبستگی بالایی با یکدیگر داشته باشند. مدل ریج با ridge_model.fit(X_train, y_train) آموزش داده می‌شود.

رگرسیون چندجمله‌ای: این مدل برای مدل‌سازی روابط غیرخطی بین ویژگی‌ها و هدف استفاده می‌شود. با استفاده از make_pipeline(PolynomialFeatures(degree=2), LinearRegression()) ویژگی‌های ورودی به صورت چندجمله‌ای درجه ۲ تبدیل می‌شوند و سپس رگرسیون خطی روی ویژگی‌های جدید انجام می‌شود. این مدل برای شبیه‌سازی روابط پیچیده‌تر و غیرخطی بین متغیرها مفید است. درجات مختلف از مقدار ۲ تا ۵ در این تمرین تست شده است اما از آنجایی که درجه ۲ کمترین خطا RMSE را ارائه می‌دهد؛ این مقدار، مقدار نهایی انتخاب شده است.

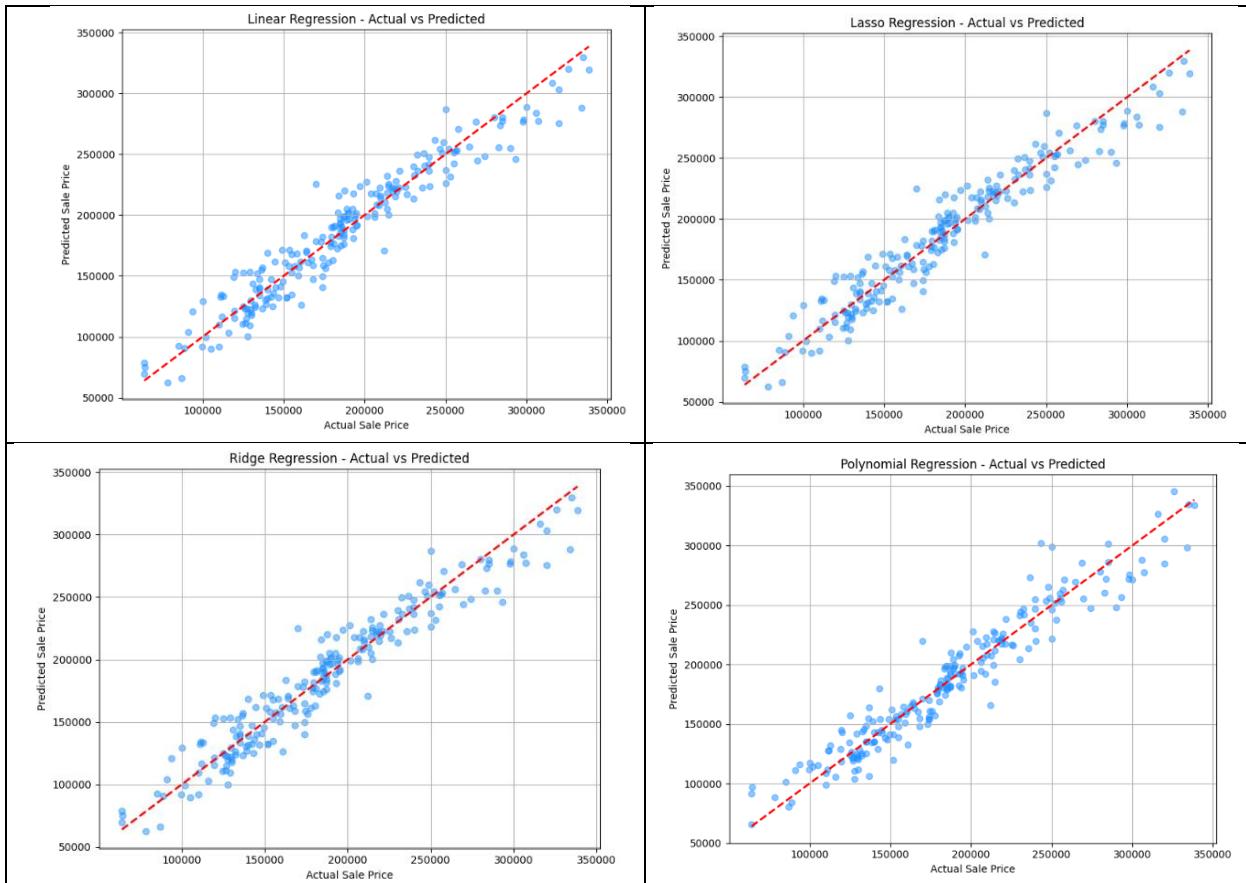
```
def plot_predictions(model, X_test, y_test, model_name="Model"):
    predictions = model.predict(X_test)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, predictions, alpha=0.5, color='dodgerblue')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
    plt.xlabel("Actual Sale Price")
    plt.ylabel("Predicted Sale Price")
    plt.title(f'{model_name} - Actual vs Predicted')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_predictions(linear_model, X_test, y_test, "Linear Regression")
plot_predictions(lasso_model, X_test, y_test, "Lasso Regression")
plot_predictions(ridge_model, X_test, y_test, "Ridge Regression")
plot_predictions(poly_model, X_test, y_test, "Polynomial Regression")
```

همچنین برای نمایش مدل های آموزش دیده از کد زیر استفاده شده است.

```
def plot_predictions(model, X_test, y_test, model_name="Model"):
    predictions = model.predict(X_test)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, predictions, alpha=0.5, color='dodgerblue')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
    plt.xlabel("Actual Sale Price")
    plt.ylabel("Predicted Sale Price")
    plt.title(f"{model_name} - Actual vs Predicted")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_predictions(linear_model, X_test, y_test, "Linear Regression")
plot_predictions(lasso_model, X_test, y_test, "Lasso Regression")
plot_predictions(ridge_model, X_test, y_test, "Ridge Regression")
plot_predictions(poly_model, X_test, y_test, "Polynomial Regression")
```



خروجی ۱۰

۱۰. بخش نهم

خطای ریشه میانگین مربعات (RMSE) یا Root Mean Squared Error) یکی از معیارهای رایج برای ارزیابی دقت مدل‌های رگرسیونی است. RMSE نشان‌دهندهٔ تفاوت میان مقادیر پیش‌بینی‌شده و مقادیر واقعی است و می‌تواند به شما بگوید که مدل چقدر از داده‌های واقعی دور است. هرچه مقدار RMSE کمتر باشد، مدل دقت بالاتری دارد.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$$

معیار R^2 یا ضریب تعیین یکی از مهم‌ترین و معروف‌ترین معیارهای ارزیابی مدل‌های رگرسیونی است. این معیار نشان‌دهندهٔ میزان تطابق مدل با داده‌های واقعی است و مقداری بین ۰ و ۱ دارد. هرچه مقدار R^2 به ۱ نزدیک‌تر باشد، مدل پیش‌بینی‌های دقیق‌تری دارد.

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

که در روابط بالا، y_i مقدار واقعی، \hat{y}_i مقدار پیش‌بینی شده و \bar{y} مقدار میانگین واقعی می‌باشد.

```
def evaluate_model(model, X_test, y_test, model_name="Model"):
    predictions = model.predict(X_test)
    r2 = model.score(X_test, y_test)
    rms = sqrt(mean_squared_error(y_test, predictions))
    print(f'{model_name} => R²: {r2:.4f}, RMS: {rms:.2f}')

evaluate_model(linear_model, X_test, y_test, "Linear Regression")
evaluate_model(lasso_model, X_test, y_test, "Lasso Regression")
evaluate_model(ridge_model, X_test, y_test, "Ridge Regression")
evaluate_model(poly_model, X_test, y_test, "Polynomial Regression")
```

```
Linear Regression => R²: 0.9267, RMS: 15777.69
Lasso Regression => R²: 0.9267, RMS: 15776.99
Ridge Regression => R²: 0.9267, RMS: 15772.91
Polynomial Regression => R²: 0.9290, RMS: 15524.22
```

که با توجه به مقادیر بدست آمده، مدل‌ها نتایج قابل قبولی دارند و بین مدل‌ها، مدل چند جمله‌ای از سایر مدل‌ها دقیق‌تر می‌باشد.

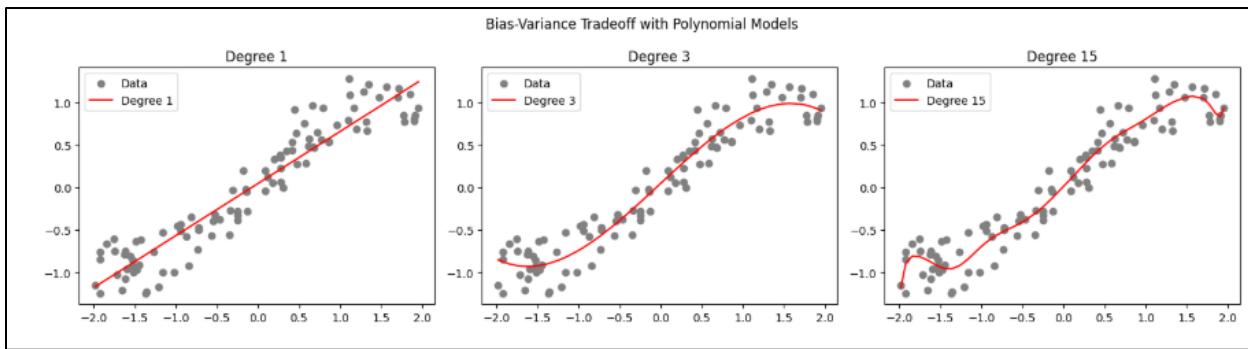
۱۱. بخش دهم

تعادل بین Bias و Variance یکی از مفاهیم اساسی در یادگیری ماشین است که تأثیر زیادی بر عملکرد مدل‌های یادگیری دارد Bias. به انحراف پیش‌بینی‌های مدل از مقادیر واقعی اشاره دارد و زمانی که مدل نتواند به درستی پیچیدگی داده‌ها را شبیه‌سازی کند، Bias بالا خواهد بود که معمولاً باعث underfitting می‌شود. از سوی دیگر، Variance نشان‌دهنده میزان نوسان در پیش‌بینی‌های مدل است و زمانی که مدل پیچیده‌ای مانند درخت تصمیم‌گیری عمیق یا شبکه عصبی به داده‌های آموزشی دقیقاً تطبیق یابد، ممکن است Variance بالا ایجاد شود که می‌تواند به overfitting منجر شود. در واقع، کاهش یکی از این دو ممکن است باعث افزایش دیگری شود، و این همان Bias-Variance Tradeoff است. به همین دلیل، برای بهبود عملکرد مدل‌های یادگیری ماشین، باید تعادلی میان این دو ایجاد کرد تا هم از underfitting جلوگیری شود و هم از overfitting. برای این منظور، از تکنیک‌هایی مانند کاهش جرم (Regularization)، کراس‌ولیدیشن و انتخاب مدل‌های مناسب استفاده می‌شود تا مدل قادر به پیش‌بینی دقیق‌تری باشد.

```
random.seed(0)
X = sort(random.rand(100, 1) * 4 - 2, axis=0)
y = sin(X).ravel() + random.randn(100) * 0.2

degrees = [1, 3, 15]
plt.figure(figsize=(15, 4))
for i, d in enumerate(degrees):
    model = make_pipeline(PolynomialFeatures(d), LinearRegression())
    model.fit(X, y)
    y_pred = model.predict(X)

    plt.subplot(1, 3, i + 1)
    plt.scatter(X, y, color='gray', label='Data')
    plt.plot(X, y_pred, color='red', label=f'Degree {d}')
    plt.title(f'Degree {d}')
    plt.legend()
plt.suptitle("Bias-Variance Tradeoff with Polynomial Models")
plt.tight_layout()
plt.show()
```



۱۲ خروجی

در این مثال، داده‌ها با افزودن نویز به تابع سینوسی تولید شده‌اند و سه مدل رگرسیون چندجمله‌ای با درجهات ۱، ۳ و ۱۵ روی آن‌ها آموزش داده شده‌اند. مدل با درجه ۱ بسیار ساده بوده و قادر به دنبال کردن الگوی واقعی داده‌ها نیست؛ این موضوع نشان‌دهنده‌ی بایاس بالا و انطباق ضعیف (Underfitting) است. مدل با درجه ۳ تطابق بهتری با داده‌ها دارد و تعادلی نسبی میان بایاس و واریانس برقرار کرده است. در مقابل، مدل با درجه ۱۵ بیش از حد به داده‌ها وابسته شده و نوسانات شدیدی از خود نشان می‌دهد؛ به‌طوری‌که حتی نویز موجود در داده‌ها را نیز دنبال می‌کند. این وضعیت بیانگر واریانس بالا و بیش‌بازش (Overfitting) است.