به نام خدا

تمرین سوم یادگیری ماشین

غزل زمانی‌نژاد

. .۱

1)

1.1) $y = \beta_0 + \varepsilon_i$

برای این مسئله regression تابع ضرر را SSE تعریف می‌کنیم:

$$L = \sum_{i=1}^{n} (y_i - \beta_0)^2$$

$$\partial L / \partial \beta_0 = -2 \sum (y_i - \beta_0) = 0 \rightarrow \sum_{i=1}^{n} y_i = n\beta_0 \rightarrow \beta_0 = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$\rightarrow \beta_0 = \frac{1}{10} \sum_{i=1}^{10} y_i = \frac{1}{10}(561) = 56.1$$

1.2) $y_i = \beta_1 x_i + \varepsilon_i$

$$L = \sum_{i=1}^{n} (y_i - \beta_1 x_i)^2 \longrightarrow \frac{\partial L}{\partial \beta_1} = -2 \sum_{i=1}^{n} (y_i - \beta_1 x_i) x_i = 0$$

$$\rightarrow \sum y_i x_i - \beta_1 x_i^2 = 0 \rightarrow \sum_{i=1}^{n} y_i x_i = \beta_1 \sum_{i=1}^{n} x_i^2 \rightarrow \beta_1 = \frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2}$$

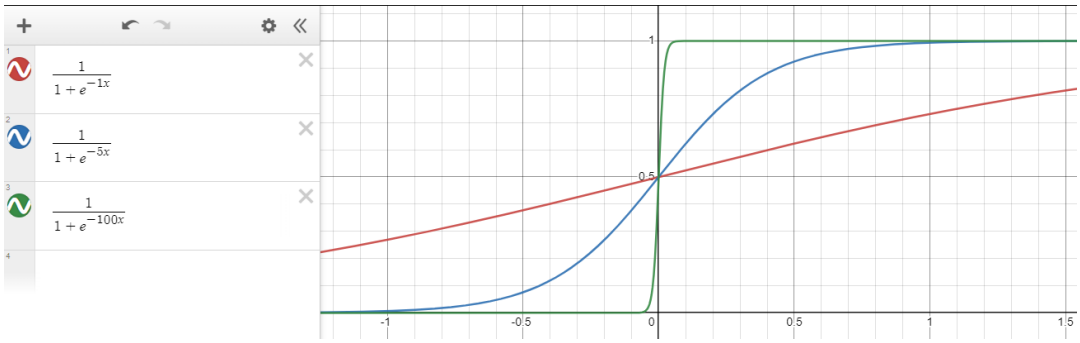
$$\rightarrow \beta_1 = \frac{\sum_{i=1}^{10} y_i x_i}{\sum_{i=1}^{10} x_i^2} = \frac{12521}{4173} \simeq 3.0004$$

1.3) $\hat{y} = 25 - 0.5(6) = 22$

خطای رگرسیون مقداری غیر صفر است یعنی برخی خونه ها را نمی‌تواند به صورت دقیق پیش بینی کند و ممکن است مقدار این خونه هم کمی با 22 متفاوت باشد

1.4) $\sigma^2 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n - k} = \frac{7}{16 - 2} = 0.5$ (k تعداد پارامترهای رگرسیون (شیب و عرض از مبدا))

۲. الف)



همانطور که در شکل مشاهده می‌شود، هرچه وزن بیشتر باشد تابع به تغییر مقدار ورودی حساس‌تر می‌شود. اگر این تغییرات به دلیل نویز در داده‌های آموزشی باشد، ممکن است مدل نویز را یاد بگیرد که منجر به کمبود تعمیم‌دهی می‌شود.

همچنین وزن‌های بزرگ اهمیت بیشتری به ویژگی‌های مرتبط می‌دهند. یعنی مدل ممکن است بیش از حد به مجموعه‌ای از ویژگی‌ها اعتماد کند، حتی اگر برای تعمیم‌دهی ضروری نباشند.

وزن‌های بزرگ به پیچیدگی بیشتری در مدل منجر می‌شوند. یک مدل بیش از حد پیچیده ممکن است به جای یادگیری الگوهای موجود، داده‌های آموزشی را حفظ کند که باعث overfit می‌شود.

ب)

2.2) $\hat{w}_{MAP} = \arg\max f(w|D) = \arg\max \sum_i \underbrace{\ln f(d_i, w)}_{p(y|x)} + \ln f(w)$

Term Ⓘ: $\log p(y_1, y_2, \dots, y_n \mid x_1, x_2, \dots, x_n) \underset{iid}{=} \log \prod_{i=1}^{n} p(y_i|x_i) =$ (سیگموید $g(x)$)

$$= \sum_{i=1}^{\hat{n}} \log p(y_i|x_i) = \sum \log \left[g(x_i)^{y_i} (1-g(x_i))^{1-y_i} \right] =$$

$$= \sum y_i \log g(x_i) + (1-y_i) \log (1-g(x_i)) =$$

$$= \sum y_i \log \sigma(w^T x_i) + (1-y_i) \log (1-\sigma(w^T x_i))$$

Term Ⓘ: $\log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{1}{2}\left(\frac{w-\mu}{\sigma}\right)^2\right) = \frac{-1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\left(\frac{w-\mu}{\sigma}\right)^2$

$w \sim N(0, I) \rightarrow \frac{-1}{2}\log(2\pi I) - \frac{1}{2}\left(\frac{w}{I}\right)^2$

$$\frac{\partial \text{Term I}}{\partial w} = \sum_{i=1}^{n} y_i \frac{x_i \sigma(w^T x_i)(1-\sigma(w^T x_i))}{\sigma(w^T x_i)} + (1-y_i) \frac{-\sigma(w^T x_i)(1-\sigma(w^T x_i)) x_i}{1-\sigma(w^T x_i)} =$$

$$= \sum_{i=1}^{n} x_i \underbrace{y_i(1-\sigma(w^T x_i))}_{y_i - y_i\sigma(w^T x_i)} - \underbrace{(1-y_i)\sigma(w^T x_i)}_{\sigma(w^T x_i) - \sigma(w^T x_i) y_i} x_i = \left[y_i - \sigma(w^T x_i) \right] x_i$$

$$\frac{\partial \text{Term II}}{\partial w} = -\frac{w}{I} \cdot \frac{1}{I} = \frac{-w}{I}$$

$$\Rightarrow \nabla_w L(w) = \sum_{i=1}^{n} \left(y_i - \sigma(w^T x_i) \right) x_i - \left(\frac{w}{I} \right) \leftarrow \text{inside regularization term باشه}$$

$$\rightarrow w_{t+1} = w_t - \eta \left[\sum_{i=1}^{n} \left(y_i - \sigma(w_t^T x_i) \right) x_i - \frac{w_t}{I} \right]$$

$$g(\lambda) = \frac{-1}{2} \sum_{i=1}^{3} \sum_{j=1}^{3} \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{i=1}^{3} \lambda_i$$

$$= \frac{-1}{2} \Big[\lambda_1 \lambda_1 y_1 y_1 x_1^T x_1 + \lambda_1 \lambda_2 y_1 y_2 x_1^T x_2 + \lambda_1 \lambda_3 y_1 y_3 x_1^T x_3 +$$

$$\lambda_2 \lambda_1 y_2 y_1 x_2^T x_1 + \lambda_2 \lambda_2 y_2 y_2 x_2^T x_2 + \lambda_2 \lambda_3 y_2 y_3 x_2^T x_3 +$$

$$\lambda_3 \lambda_1 y_3 y_1 x_3^T x_1 + \lambda_3 \lambda_2 y_3 y_2 x_3^T x_2 + \lambda_3 \lambda_3 y_3 y_3 x_3^T x_3 \Big]$$

$$+ \lambda_1 + \lambda_2 + \lambda_3 =$$

$$= \frac{-1}{2} \Big[\lambda_1^2 \cdot 1 \cdot \overbrace{[1,2]\begin{bmatrix}1\\2\end{bmatrix}}^{5} + \lambda_1 \lambda_2 \cdot 1 \cdot \overbrace{[1,2]\begin{bmatrix}-1\\2\end{bmatrix}}^{3} + \lambda_1 \lambda_3 (-1) \overbrace{[1,2]\begin{bmatrix}-1\\-2\end{bmatrix}}^{-5} +$$

$$\lambda_2 \lambda_1 \cdot 1 \overbrace{[-1,2]\begin{bmatrix}1\\2\end{bmatrix}}^{3} + \lambda_2^2 \cdot 1 \overbrace{[-1,2]\begin{bmatrix}-1\\2\end{bmatrix}}^{5} + \lambda_2 \lambda_3 (-1) \overbrace{[-1,2]\begin{bmatrix}-1\\-2\end{bmatrix}}^{-3} +$$

$$\lambda_3 \lambda_1 (-1) \underbrace{[-1,-2]\begin{bmatrix}1\\2\end{bmatrix}}_{-5} + \lambda_3 \lambda_2 (-1) \underbrace{[-1,-2]\begin{bmatrix}-1\\2\end{bmatrix}}_{-3} + \lambda_3^2 \cdot 1 \underbrace{[-1,-2]\begin{bmatrix}-1\\-2\end{bmatrix}}_{5} \Big]$$

$$+ \lambda_1 + \lambda_2 + \lambda_3$$

$$g(\lambda) = \frac{-1}{2} \Big[5\lambda_1^2 + 6\lambda_1\lambda_2 + 10\lambda_1\lambda_3 + 5\lambda_2^2 + 6\lambda_2\lambda_3 + 5\lambda_3^2 \Big] + \lambda_1 + \lambda_2 + \lambda_3$$

$$\max_{\lambda_1, \lambda_2, \lambda_3} g(\lambda) \quad \text{s.t.} \begin{cases} \lambda_i \geq 0 & i = 1,2,3 \\ \sum_{i=1}^{3} \lambda_i y_i = -\lambda_1 - \lambda_2 + \lambda_3 = 0 \end{cases}$$

$$\partial g / \partial \lambda_1 = \frac{-1}{2} \big[10\lambda_1 + 6\lambda_2 + 10\lambda_3 \big] + 1 = -5\lambda_1 - 3\lambda_2 - 5\lambda_3 + 1 = 0$$

$$\rightarrow 5\lambda_1 + 3\lambda_2 + 5\lambda_3 = 1$$

$$\partial g / \partial \lambda_2 = \frac{-1}{2} \big[6\lambda_1 + 10\lambda_2 + 6\lambda_3 \big] + 1 = -3\lambda_1 - 5\lambda_2 - 3\lambda_3 + 1 = 0$$

$$\rightarrow 3\lambda_1 + 5\lambda_2 + 3\lambda_3 = 1$$

$$\partial g / \partial \lambda_3 = \frac{-1}{2} \big[10\lambda_1 + 6\lambda_2 + 10\lambda_3 \big] + 1 = -5\lambda_1 - 3\lambda_2 - 5\lambda_3 + 1 = 0$$

$$\rightarrow 5\lambda_1 + 3\lambda_2 + 5\lambda_3 = 1$$

$$\Rightarrow \begin{bmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \lambda_1 = \frac{1}{8} - \lambda_3 \quad , \quad \lambda_2 = \frac{1}{8}$$

$$w = \sum_{i=1}^{3} \lambda_i y_i x_i = (\frac{1}{8} - \lambda_3)(-1)(1,2) + \frac{1}{8}(-1)(-1,2) + \lambda_3 \cdot 1(-1,-2)$$

$$= (\lambda_3 - \frac{1}{8}, 2\lambda_3 - \frac{1}{4}) + (\frac{1}{8}, \frac{-1}{4}) + (-\lambda_3, -2\lambda_3)$$

$$= (0, -\frac{1}{2})$$

$b \rightarrow x_3$ استفاده از $y_3 = +1$ $\longrightarrow$ $w^T x_3 + b = 1 \rightarrow (0, \frac{-1}{2})(\underbrace{-1}_{-2}) + b = 1 \rightarrow b = 0$

معادله خط $\Longrightarrow w^T x + b = 0 \rightarrow (\frac{0}{\frac{-1}{2}}) x + 0 = 0 \rightarrow \frac{-1}{2} x_2 = 0$

۴. الف) Kernelها در SVM برای تبدیل فضای ویژگی‌های ورودی به یک فضای با ابعاد بالاتر مورد استفاده قرار می‌گیرند. این تبدیلات به SVM امکان را می‌دهد که در فضای با ابعاد بالاتر، مسائل غیرخطی را به شکل خطی حل کند. مفهوم اصلی این است که اگر داده‌ها در فضای ویژگی به شکل خطی جداپذیر نباشند، می‌توان با افزودن بعد جدید به فضا، داده‌ها را به شکل خطی جداپذیر کرد. کرنل تابعی است که ورودی را به یک فضای با ابعاد بالاتر تبدیل می‌کند، بدون این‌که نیاز به محاسبه و ذخیره فضای جدید باشد و این مورد از مزایای کرنل است. کافیست حاصل ضرب نقطه‌ای دو ویژگی را در فضای دیگر بدانیم. حل مسئله به صورت خطی هم مزیت دیگر آن است.

ب)

4.2) از آن جایی که k_1 و k_2 دو کرنل معتبر هستند:

$$k_1(x,x') = \sum_i \phi^1(x_i)\phi^1(x'_i) \quad , \quad k_2(x,x') = \sum_i \phi^2(x_i)\phi^2(x'_i)$$

$$k(x,x') = \left(\sum_i \phi^1(x_i)\phi^1(x'_i)\right) \times \left(\sum_j \phi^2(x_j)\phi^2(x'_j)\right) =$$

$$= \sum_i \sum_j \phi^1(x_i)\phi^2(x_j)\phi^1(x'_i)\phi^2(x'_j)$$

$$\xrightarrow{\phi_k(z) = \phi^1(z_i)\phi^2(z_j)} \sum_k \phi(x_k)\phi(x'_k) = k(x,x')$$

۵. .

۵,۱) ابتدا دیتاست را لود می‌کنیم. این داده شامل ۸ ستون و ۵۰۰ سطر است. مقادیر آماری ستون‌های عددی در جدول مشاهده می‌شود:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Email                 500 non-null    object 
 1   Address               500 non-null    object 
 2   Avatar                500 non-null    object 
 3   Avg. Session Length   500 non-null    float64
 4   Time on App           500 non-null    float64
 5   Time on Website       500 non-null    float64
 6   Length of Membership  500 non-null    float64
 7   Yearly Amount Spent   500 non-null    float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

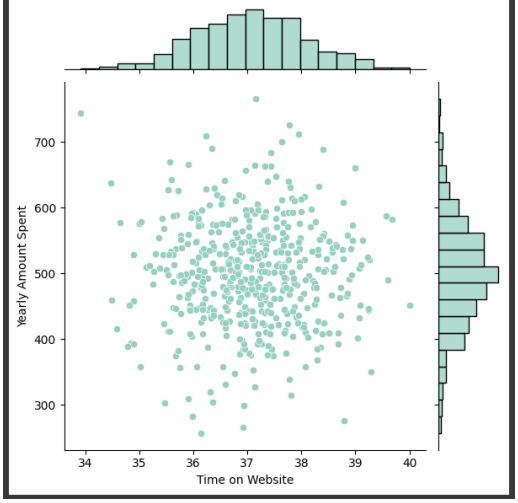
```
1 df.describe()
```

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

۵,۲) پلات joint دو ویژگی خواسته شده:

```
1 import seaborn as sns
2
3 sns.set_palette('GnBu_d')
4 sns.jointplot(data=df, y='Yearly Amount Spent', x='Time on Website')
```

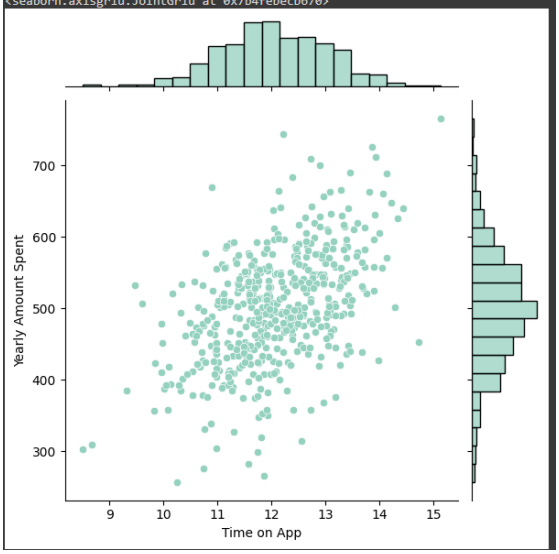
```
<seaborn.axisgrid.JointGrid at 0x7b501e4fa020>
```



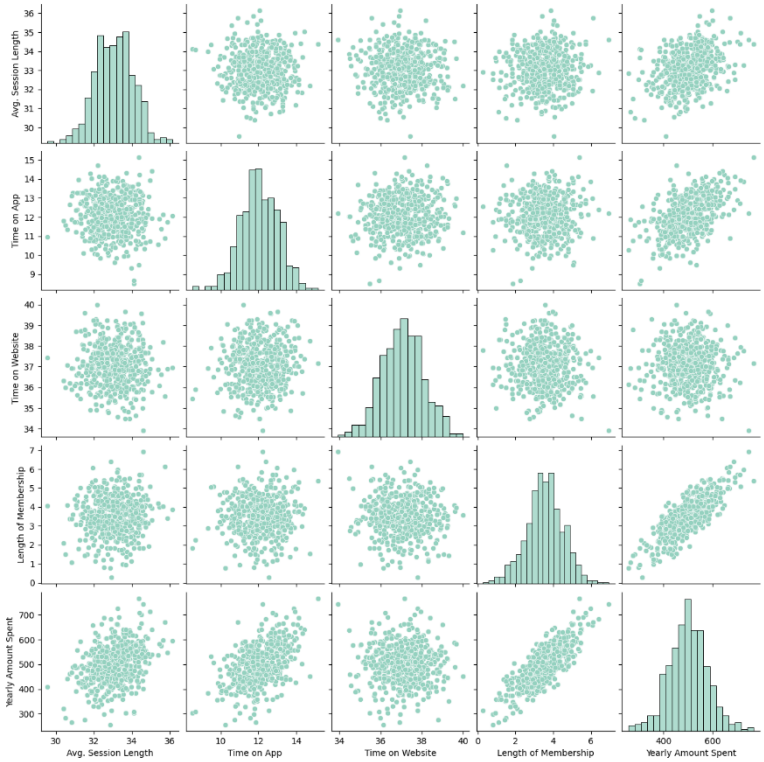
(۵,۳

```
import seaborn as sns

sns.set_palette('GnBu_d')
sns.jointplot(data=df, y='Yearly Amount Spent', x='Time on App')
```

```
<seaborn.axisgrid.JointGrid at 0x7b4febecb670>
```



۵,۴) پلات رسم شده:



محور عمودی در نمودارهای ردیف آخر Yearly Amount Spent است. به نظر می‌رسد فیچر Length of Membership بیشترین تاثیر را روی میزان خرید دارد و با کم و زیاد شدن طول عضویت، مقدار خرج سالیانه تغییر بیشتری داشته.

۵،۵) ستون‌های عددی را به عنوان ویژگی در نظر می‌گیریم. البته ممکن است آدرس هم در میزان خرید سالیانه تاثیر داشته باشد ولی دسته‌بندی آن با توجه به تعداد کم داده دشوار است پس از آن صرف نظر می‌کنیم. سپس مقادیر ویژگی‌ها را نرمالیزه می‌کنیم زیرا مقادیر ممکن برای ویژگی‌های مختلف در بازه‌های مختلف قرار دارند و با نرمالیزه کردن همه مقادیر را به یک مقیاس می‌بریم.

```
print(df.columns)
# drop Email, Address and Avatar. Wouldn't help as feature
X = df.iloc[:, 3:-1]
y = df['Yearly Amount Spent']

X
```

```
Index(['Email', 'Address', 'Avatar', 'Avg. Session Length', 'Time on App',
       'Time on Website', 'Length of Membership', 'Yearly Amount Spent'],
      dtype='object')
```

	Avg. Session Length	Time on App	Time on Website	Length of Membership
0	34.497268	12.655651	39.577668	4.082621
1	31.926272	11.109461	37.268959	2.664034
2	33.000915	11.330278	37.110597	4.104543
3	34.305557	13.717514	36.721283	3.120179
4	33.330673	12.795189	37.536653	4.446308
...	...	...	...	...
495	33.237660	13.566160	36.417985	3.746573
496	34.702529	11.695736	37.190268	3.576526
497	32.646777	11.499409	38.332576	4.958264
498	33.322501	12.391423	36.840086	2.336485
499	33.715981	12.418808	35.771016	2.735160

500 rows × 4 columns

```
from sklearn.preprocessing import StandardScaler

# normalize each column
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

۵،۶)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

۵،۷) مدل رگرسیون را آموزش می‌دهیم. دقت آن روی داده آموزشی حدود ۹۸ درصد است.

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X_train, y_train)
reg.score(X_train, y_train)
```

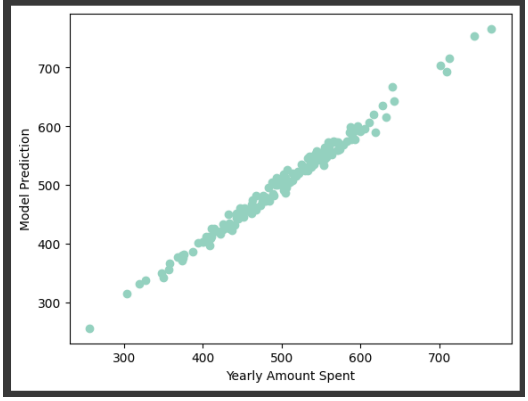
```
0.9817562058732432
```

۵،۸)

```
pred = reg.predict(X_test)
```

(۵,۹

```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(y_test, pred)
4 plt.xlabel("Yearly Amount Spent")
5 plt.ylabel("Model Prediction")
6 plt.show()
```



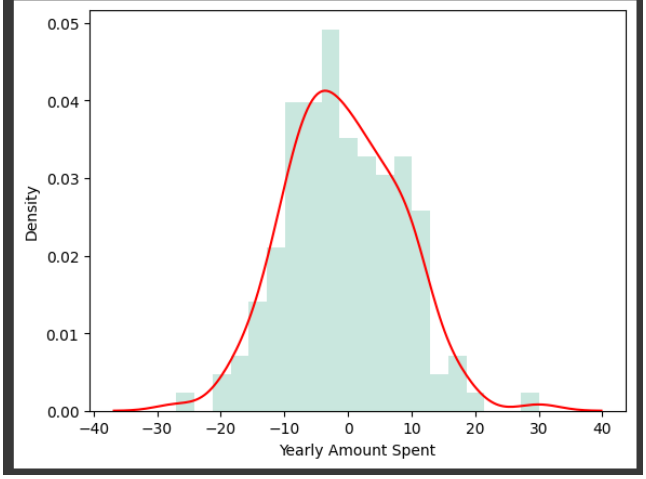
(۵,۱۰

```
1 import numpy as np
2
3 mae = np.mean(np.abs(y_test - pred))
4 mse = np.mean((y_test - pred)**2)
5 root_mse = np.sqrt(mse)
6
7 print('MAE:', mae, '\nMSE:', mse, '\nRoot MSE:', root_mse)
```

```
MAE: 7.228148653430832
MSE: 79.81305165097451
Root MSE: 8.933815066978637
```

۵,۱۱) هیستوگرام و نمودار kde را در یک پلات رسم می‌کنیم. اگر مدل به خوبی فیت شده باشد، خطاهای پیش‌بینی باید به طور مساوی در اطراف مقدار واقعی توزیع شوند. یعنی خطاهای مثبت و منفی باید به طور مساوی توزیع شوند. علاوه بر این، شدت خطاها باید به طور مساوی توزیع شوند. میانگین خطا هم باید صفر شود که اینها توصیفی از توزیع نرمال است و در شکل هم نمودار تفاضل مقادیر به نرمال نزدیک است.

```
difference = y_test - pred

plt.hist(difference, bins=20, alpha=0.5, density=True)
sns.kdeplot(difference, color='red')
plt.show()
```



۵،۱۲) مدل برای هر یک از ویژگی‌ها وزنی در نظر گرفته است.

```
reg.coef_
```

```
array([25.76252659, 38.32855202,  0.19220992, 61.17355707])
```

در رگرسیون خطی، coef، اهمیت هر فیچر را نشان می‌دهد. هر چه مقدار coef یک فیچر بیشتر باشد، آن فیچر در پیش‌بینی خروجی مدل تاثیر بیشتری دارد. اگر داده‌ها نرمالیزه نباشند، ممکن است ضرایب به طور غیرواقعی بزرگ یا کوچک شوند و منجر به پیش‌بینی اشتباه شود. این دلیل است که فیچرهایی که دامنه بزرگتری دارند، تاثیر بیشتری بر مدل خواهند داشت.

۵،۱۳) ضریب ویژگی آخر یعنی Length of Membership از همه بزرگتر است پس یعنی این ویژگی از اهمیت بیشتری برخوردار است. بهتر از سرمایه‌گذاری روی این طول عضویت مشتریان انجام شود.

۶. .

۶,۱) این دیتاست از ۱۰ ستون و ۱۰۰۰ سطر تشکیل شده‌است.

```
1 import pandas as pd
2
3 df = pd.read_csv('./advertising.csv')
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object
 5   City                      1000 non-null   object
 6   Male                      1000 non-null   int64
 7   Country                   1000 non-null   object
 8   Timestamp                 1000 non-null   object
 9   Clicked on Ad             1000 non-null   int64
dtypes: float64(3), int64(3), object(4)
memory usage: 78.2+ KB
```

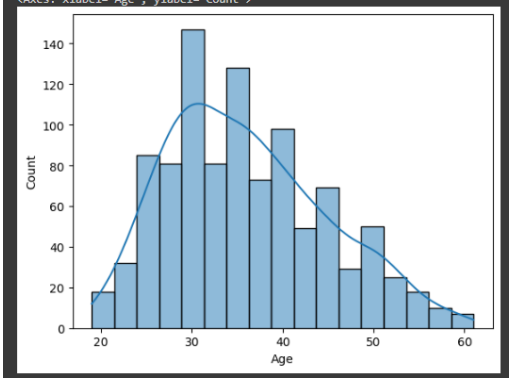
```
1 df.describe()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.000000	1.00000

۶,۲)

```
1 import seaborn as sns
2
3 sns.histplot(df['Age'], kde=True)
```

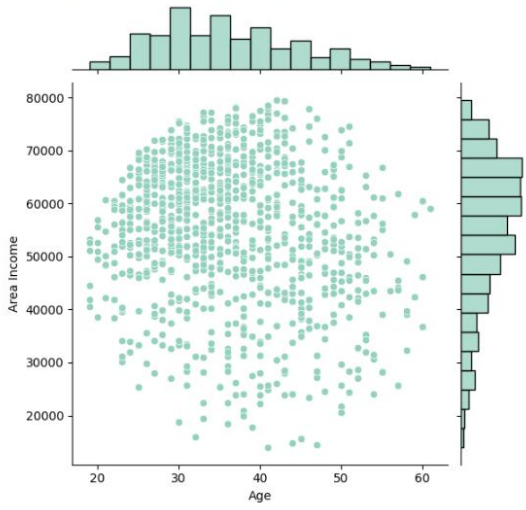
```
<Axes: xlabel='Age', ylabel='Count'>
```



(۶,۳

```
1 sns.set_palette('GnBu_d')
2 sns.jointplot(data=df, y='Area Income', x='Age')
```

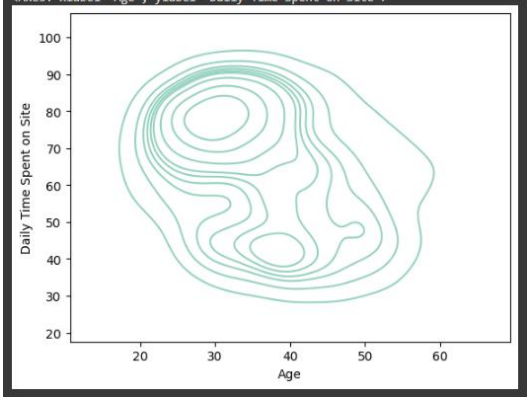
```
<seaborn.axisgrid.JointGrid at 0x786309de8f10>
```



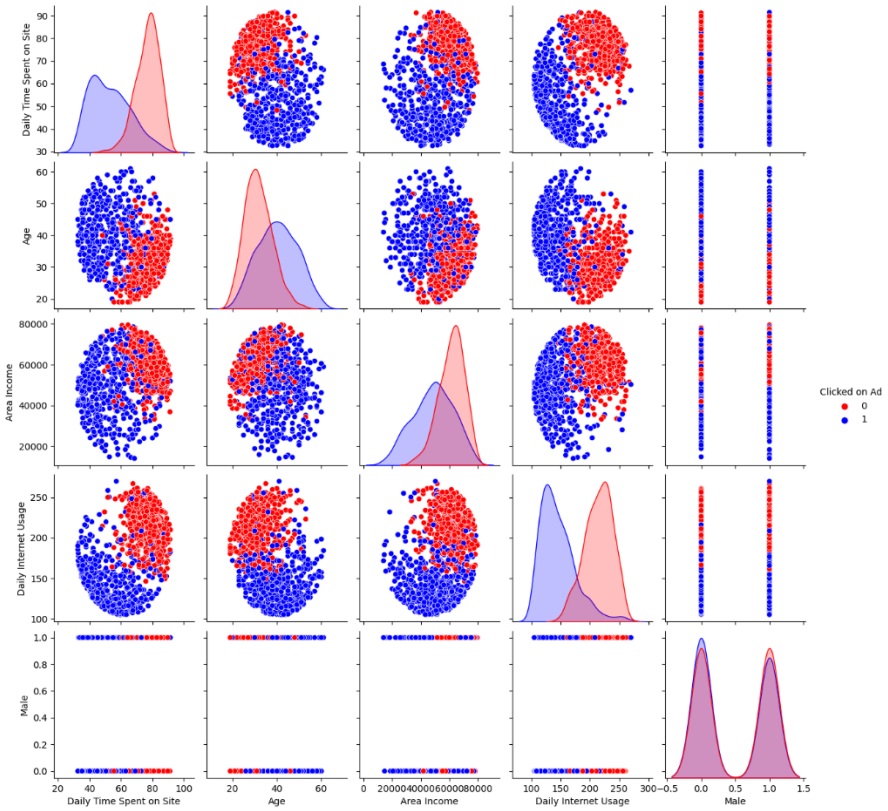
(۶,۴

```
1 sns.kdeplot(data=df, y='Daily Time Spent on Site', x='Age')
```

```
<Axes: xlabel='Age', ylabel='Daily Time Spent on Site'>
```



(۶,۵



۶,۶) افرادی که حوالی ۳۰ سال دارند بیشترین کلیک را روی تبلیغات کرده‌اند. میان سن و درآمد افراد تا حدی correlation وجود دارد. بیشتر جمعیت این دیتاست، بین ۳۰ تا ۴۰ سال سن دارند. همچنین از کانتورهای موجود در پلات kde می‌توان یافت که بیشتر افراد بین ۳۰ تا ۴۰ سال بیشترین زمان را در سایت صرف کرده‌اند و این میزان بیشتر بین ۷۰ تا ۸۰ دقیقه بوده‌است. به طور کلی با توجه به pair plot، با کمتر شدن زمان صرف شده در سایت، بالاتر رفتن سن، کمتر بودن درآمد، و مصرف کمتر اینترنت در طول روز، تعداد کلیک بر روی تبلیغ بیشتر شده‌است.

۶,۷) ۵ ستون: میزان زمان صرف شده در سایت به طور روزانه ، سن، درآمد، میزان اینترنت مصرفی به طور روزانه و مرد بودن را به عنوان ویژگی داده برای مسئله طبقه‌بندی در نظر می‌گیریم.
سپس برای اینکه مقادیر عددی همگی در یک مقیاس قرار گیرند، داده‌ها را نرمالیزه می‌کنیم. ۳۰ درصد از داده را به عنوان تست جدا می‌کنیم.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

X = df[['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Male']]
y = df['Clicked on Ad']

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape)
```

```
(700, 5)
```

۶,۸) یک کلاس logistic regression تعریف می‌کنیم. برای حذف بایاس و اضافه کردن آن به بردار وزن، در ابتدا یک بعد به ورودی اضافه می‌کنیم. سپس بردار وزن را با ابعاد ۱*(d+۱) با ۰ مقداردهی اولیه می‌کنیم. توابع مورد نیاز را تعریف می‌کنیم:

```python
class LogisticReg:
    def __init__(self, X, y, epochs=1000, alpha=0.01):
        # add one more dim to input (bias)
        self.X = np.hstack((np.ones((X.shape[0], 1)), X))
        self.y = y

        # initialize weights
        self.W = np.zeros((self.X.shape[1], 1))

        self.epochs = epochs
        self.alpha = alpha

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def forward(self, X):
        z = np.dot(X, self.W)
        return self.sigmoid(z)

    def loss(self, y_hat, y):
        l = -1 * np.average(y * np.log(y_hat) + (1-y) * np.log(1-y_hat))
        return l

    def grads(self, X, y_hat, y):
        # round L/round w = x (y_hat - y)
        return np.dot(X.T, np.expand_dims(y_hat-y, axis=1)) / len(y)
```

در تابع train به تعداد ایپاک باید مدل را آموزش دهیم. برای این کار ابتدا یک مرحله forward propagation انجام می‌دهیم و بعد میزان خطای cross entropy را حساب می‌کنیم. سپس گرادیان را محاسبه و بردار وزن را آپدیت می‌کنیم.

از تابع predict برای پیش‌بینی داده تست استفاده می‌کنیم. یک مرحله forward را انجام می‌دهیم و به داده با توجه به خروجی و ترشهولد (در اینجا ۰,۵) برچسب می‌زنیم.

```python
    def train(self):

        for e in range(self.epochs):
            # forward propagation
            y_hat = np.squeeze(self.forward(self.X))
            # compute loss
            l = self.loss(y_hat, self.y)
            # compute gradients
            dw = self.grads(self.X, y_hat, self.y)
            # update parameters
            self.W -= self.alpha * dw

            if e % 100 == 0:
                print(f'Epoch {e}, Loss:', "{:.3f}".format(l))

    def predict(self, X):
        # add one more dimension
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        probs = self.forward(X)
        preds = (probs >= 0.5).astype(int)
        return preds

```

مدل را در ۱۰۰۰ ایپاک و با learning rate ۰,۰۱ آموزش می‌دهیم. دقت مدل روی داده آموزش حدود ۹۷٪ است.

```python
model = LogisticReg(X_train, y_train)
model.train()
```

```
Epoch 0, Loss: 0.693
Epoch 100, Loss: 0.424
Epoch 200, Loss: 0.311
Epoch 300, Loss: 0.253
Epoch 400, Loss: 0.219
Epoch 500, Loss: 0.196
Epoch 600, Loss: 0.180
Epoch 700, Loss: 0.168
Epoch 800, Loss: 0.159
Epoch 900, Loss: 0.151
```

```python
from sklearn.metrics import accuracy_score, classification_report,

pred_train = model.predict(X_train)
acc_train = accuracy_score(y_train, pred_train)
print("Train Accuracy", "{:.3f}".format(acc_train))
```

```
Train Accuracy 0.969
```

۶,۹)

```python
pred_test = model.predict(X_test)
pred_test.shape
```

```
(300, 1)
```

۶,۱۰) میزان دقت مدل روی داده آزمون ۹۵٪ است. مطابق confusion matrix، ۲ داده برچسب واقعی ۰ داشتند و به اشتباه ۱ پیش‌بینی شدند. در مقابل ۱۳ داده برچسب واقعی ۱ داشتند و به اشتباه ۰ پیش‌بینی شدند.

```python
pred_test = model.predict(X_test)
acc_test = accuracy_score(y_test, pred_test)
print("Test Accuracy", "{:.3f}".format(acc_test))
```

```
Test Accuracy 0.950
```

```python
print(confusion_matrix(y_test, pred_test))
```

```
[[144   2]
 [ 13 141]]
```

```python
print(classification_report(y_test, pred_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       146
           1       0.99      0.92      0.95       154

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

۷. ابتدا داده را لود می‌کنیم و توزیع آن را بررسی می‌کنیم. ۸۶٪ از داده ham است.

```
import pandas as pd

df = pd.read_csv('./spamSMS.csv', encoding='ISO-8859-1')
```

```
data = df['v2']
labels = df['v1']

print("data count:", len(data))
```

```
data count: 5572
```

```
classes = pd.unique(labels)

c0 = len(labels[labels == classes[0]])
c1 = len(labels[labels == classes[1]])
print(f'{classes[0]} count: {c0} ({c0/ len(labels)}%)')
print(f'{classes[1]} count: {c1} ({c1/ len(labels)}%)')
```

```
ham count: 4825 (0.8659368269921034%)
spam count: 747 (0.1340631730789664%)
```

از count vectorizer برای استخراج ویژگی استفاده می‌کنیم. این تابع تعداد تکرار هر کلمه موجود در vocabulary را می‌شمارد و در یک دیکشنری ذخیره می‌کند.

داده را به دو بخش آموزش و آزمون تقسیم می‌کنیم.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)
```

```
print("train data shape:", X_train.shape)
print("test data shape:", X_test.shape)
```

```
train data shape: (3900, 8672)
test data shape: (1672, 8672)
```

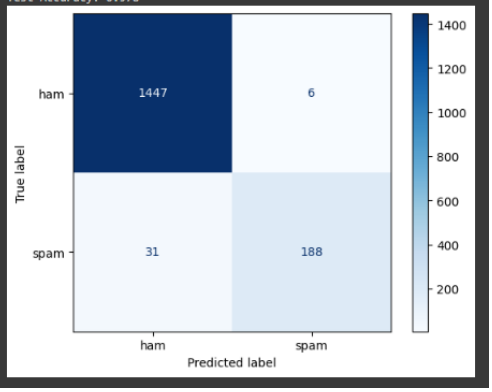
Grid Search: در این روش فضای ابرپارامترهای ممکن مشخص شده و تمام ترکیب‌های ممکن آنها بررسی می‌شوند. با استفاده از grid search کل فضای حالت امتحان می‌شود و می‌توانیم حالت بهینه را پیدا کنیم (در صورتی که در محدود جستجو باشد). اما بدلیل بررسی کل فضای جستجو، این روش زمان‌بر است و برای فضای حالت بزرگ (حالات زیاد برای ابرپارامترهای زیاد) مناسب نیست.

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

parameters = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': [0.1, 1, 10]}
svc = SVC()
grid_search_svm = GridSearchCV(svc, parameters)
grid_search_svm.fit(X_train, y_train)

print(sorted(grid_search_svm.cv_results_.keys()))
```

پس از آموزش مدل با ابرپارامترهای گوناگون به روش grid به نتایج زیر رسیدیم:

```python
def evaluate(model, X_test, y_test):
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("Test Accuracy:", "{:.3f}".format(acc))

    disp = ConfusionMatrixDisplay.from_estimator(
        model,
        X_test,
        y_test,
        display_labels=classes,
        cmap=plt.cm.Blues,
        )

    plt.show()

best_grid_param = grid_search_svm.best_params_
print("grid search best parameters:", best_grid_param)

best_grid_model = grid_search_svm.best_estimator_
print("Best model evaluation")
evaluate(best_grid_model, X_test, y_test)
```

```
grid search best parameters: {'C': 1, 'gamma': 0.1, 'kernel': 'linear'}
Best model evaluation
Test Accuracy: 0.978
```



در این روش svm با کرنل خطی و گاما ۰٫۱ و c=۱ بهترین نتیجه را بدست آورد و به دقت ۹۷٪ روی داده آزمون رسید (نتایج سایر مدل‌ها نیز در نوت‌بوک موجود است).

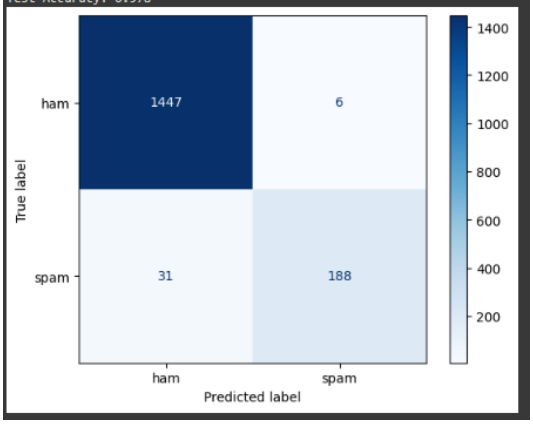
Random Search: در این روش تعدادی نمونه تصادفی از فضای ابرپارامترها انتخاب می‌شوند و مدل‌ها با آنها آموزش می‌بینند. این روش می‌تواند در فضای ابرپارامترهای پیچیده و تعداد زیاد ابعاد مفید باشد اما بدلیل اینکه تمامی فضای جستجو مورد بررسی قرار نمی‌گیرد، ممکن است به حالت بهینه نرسیم. در این سوال ابرپارامترها را برای ۱۰ حالت تصادفی انتخاب می‌کنیم و مدل آموزش می‌دهیم.

```python
svm_model = SVC()

parameters = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': [0.1, 1, 10]}
random_search_svm = RandomizedSearchCV(svm_model, param_distributions=parameters, n_iter=10)
random_search_svm.fit(X_train, y_train)
```

پس از آموزش مدل با ابرپارامترهای گوناگون به روش random به نتایج زیر رسیدیم:

```python
best_random_param = random_search_svm.best_params_
print("random search best parameters:", best_random_param)

best_random_model = random_search_svm.best_estimator_
print("Best model evaluation")
evaluate(best_random_model, X_test, y_test)
```

```
random search best parameters: {'kernel': 'linear', 'gamma': 10, 'C': 1}
Best model evaluation
Test Accuracy: 0.978
```



در این روش svm با کرنل خطی و گاما ۱۰ و c=۱ بهترین نتیجه را بدست آورد و به دقت ۹۷٪ روی داده آزمون رسید (نتایج سایر مدل‌ها نیز در نوت‌بوک موجود است).

مقایسه دو روش: grid از نظر زمانی بسیار بیشتر از random طول می‌کشد. از لحاظ کارایی grid می‌تواند به حالت بهینه برسد در صورتی که ممکن است random به بهینه نرسد.

۸. ابتدا داده را لود می‌کنیم:

```
import pandas as pd

df = pd.read_csv('./housePricing.csv')
df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	8	2007	WD	Normal	175000
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	2	2010	WD	Normal	210000
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	2500	5	2010	WD	Normal	266500
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal	142125
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2008	WD	Normal	147500

1460 rows × 81 columns

سپس بر روی آن پیش‌پردازش‌های زیر را انجام می‌دهیم: ستون‌هایی که بیشتر از ۴۰٪ داده‌شان مقدار ندارند و nan هستند را حذف می‌کنیم. برای سایر ستون‌ها، اگر متنی باشند مقدار nan را به unknown و اگر عددی باشند با مقدار میانگین آن ستون جا به جا می‌کنیم.

```
# count the number of missing values of each column
nan = list(df.isna().sum())

# find the columns' names which need to be replaced or dropped
replaced_col_text = []
replaced_col_number = []
dropped_col = []

for idx, name in enumerate(df.columns):
    if nan[idx] == 0:
        continue
    if nan[idx] > len(df) * 0.4:
        dropped_col.append(name)
    # find text columns which have nan value inside
    elif df[name].dtype == object:
        replaced_col_text.append(name)
    # find digit columns which have nan value
    else: replaced_col_number.append(name)

print("text columns with nan:", replaced_col_text)
print("number columns with nan:", replaced_col_number)
print("dropping columns:", dropped_col)

# replace missing value of a text column with 'unknown'
df[replaced_col_text] = df[replaced_col_text].fillna(value='unknown')
# # replace missing value of a digit column with its column average
df[replaced_col_number] = df[replaced_col_number].fillna(value=df[replaced_col_number].mean())
# drop columns with more than 40% missing value
df_dropped = df.drop(columns=dropped_col, axis=1)

# check if any column has missing value
for col in df_dropped.columns:
    if df_dropped[col].isna().any():
        print(col, "has na value")
```

```
text columns with nan: ['MasVnrType', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']
number columns with nan: ['LotFrontage', 'MasVnrArea', 'GarageYrBlt']
dropping columns: ['Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature']
```

در خروجی نام ستون‌هایی که مورد تغییر یا حذف قرار گرفته‌اند مشاهده می‌شود.

سپس ستون‌هایی که مقدار غیرعددی دارند، باید مقادیر آنها را با یک مقدار عددی تعویض کنیم. برای این کار از one hot encoder کتابخانه sklearn استفاده می‌کنیم. برای این کار ستون مربوطه حذف می‌شود و چند ستون (تعداد بستگی به مقادیر منحصر به‌فرد آن ستون دارد) اضافه می‌شود و با روش one hot مقادیر را ذخیره می‌کنیم.

```python
from sklearn.preprocessing import OneHotEncoder

text_columns = df_dropped.select_dtypes(include=['object']).columns
# apply one-hot encoding on text columns
one_hot = pd.get_dummies(df_dropped[text_columns])
# # Drop the original text columns and add numeric ones
new_df = pd.concat([df_dropped.drop(text_columns, axis=1), one_hot], axis=1)
new_df
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	SaleCondition_Abnorml	S
0	1	60	65.0	8450	7	5	2003	2003	196.0	706	...	0	0	0	1	0	
1	2	20	80.0	9600	6	8	1976	1976	0.0	978	...	0	0	0	1	0	
2	3	60	68.0	11250	7	5	2001	2002	162.0	486	...	0	0	0	1	0	
3	4	70	60.0	9550	7	5	1915	1970	0.0	216	...	0	0	0	1	1	
4	5	60	84.0	14260	8	5	2000	2000	350.0	655	...	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	62.0	7917	6	5	1999	2000	0.0	0	...	0	0	0	1	0	
1456	1457	20	85.0	13175	6	6	1978	1988	119.0	790	...	0	0	0	1	0	
1457	1458	70	66.0	9042	7	9	1941	2006	0.0	275	...	0	0	0	1	0	
1458	1459	20	68.0	9717	5	6	1950	1996	0.0	49	...	0	0	0	1	0	
1459	1460	20	75.0	9937	5	6	1965	1965	0.0	830	...	0	0	0	1	0	

1460 rows × 283 columns

بدین ترتیب بعد از این کار، تعداد ستون‌ها به ۲۸۳ افزایش می‌یابد.

بعد ویژگی‌ها و همچنان قیمت خانه را نرمالیزه می‌کنیم.

```python
from sklearn.preprocessing import StandardScaler

X = new_df.drop('SalePrice', axis=1)  # features
y = new_df['SalePrice']  # labels

numeric_columns = X.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X[numeric_columns] = scaler.fit_transform(X[numeric_columns])
scaler = StandardScaler()
y = scaler.fit_transform(y.values.reshape(-1, 1)).ravel()
```

۲۰ ویژگی برتر را انتخاب می‌کنیم و داده را به سه بخش آموزش، اعتبارسنجی و آزمون تقسیم می‌کنیم.

```python
from sklearn.feature_selection import SelectKBest, f_regression

k_best = 20
selector = SelectKBest(score_func=f_regression, k=k_best)
X_selected = selector.fit_transform(X, y)
```

```python
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(X_selected, y, test_size=0.3, random_state=4)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=4)

print("train features shape:", X_train.shape)
print("train labels shape:", y_train.shape)
```

```
train features shape: (1022, 20)
train labels shape: (1022,)
```

پس از تمیز کردن داده، مدل رگرسیون را آموزش می‌دهیم و نتیجه را بر روی داده‌ها گزارش می‌کنیم.

```python
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# SVM Regression model
model = SVR()
model.fit(X_train, y_train)

# Make predictions on training, validation, and test sets
train_pred = model.predict(X_train)
val_pred = model.predict(X_val)
test_pred = model.predict(X_test)

# calcualate loss on each split of data
loss_train = mean_squared_error(y_train, train_pred)
loss_val = mean_squared_error(y_val, val_pred)
loss_test = mean_squared_error(y_test, test_pred)

print('Loss on train data:', "{:.3f}".format(loss_train))
print('Loss on val data:', "{:.3f}".format(loss_val))
print('Loss on test data:', "{:.3f}".format(loss_test))
```

```
Loss on train data: 0.129
Loss on val data: 0.090
Loss on test data: 0.136
```

مطابق انتظار، میزان خطا روی داده آزمون از همه بیشتر است اما تفاوت چندانی با خطای آموزش ندارد و مدل دچار overfit نشده است.