گزارش کار

پروژه درس هوش مصنوعی

محمد ابراهیم زاده ۸۱۰۶۰۱۰۲۳

استاد: دکتر مسعود شریعت پناهی

بهار ۱۴۰۲

در ابتدا کتابخانه هایی که جهت استفاده در سوال اول مورد استفاده قرار خواهند گرفت فراخوانی میکنیم و سپس با استفاده از کتابخانه pandas فایل csv را وارد میکنیم:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
dataset = pd.read_csv(r'C:\Users\Nima\Desktop\AI Project 2/breast_cancer.csv')
```

الف) ابتدا نشان دهید دادگانی که در اختیار شما قرار گرفته دارای نقصان، پرتی داده و عیوبی از این دست نیست، سپس عددهای نشان دهندهٔ نوع تومور را از ۲ و ۴ به ترتیب به ۰ و ۱ تغییر دهید. در پایان در صورت نیاز ویژگی‌های ۹ گانه را نرمال‌سازی کنید. (راهنمایی: برای نرمال‌سازی داده‌ها می‌توانید از روش های مختلفی مانند اسکیل کردن تمامی داده‌ها بین ۰ و ۱ استفاده کنید.)

پاسخ: طبق تصویر با استفاده از کد describe داده های وارد شده را مورد بررسی قرار میدهیم:

```
dataset.describe()
```

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
count	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000
mean	4.442167	3.150805	3.215227	2.830161	3.234261	3.544656	3.445095	2.869693	1.603221	2.699854
std	2.820761	3.065145	2.988581	2.864562	2.223085	3.643857	2.449697	3.052666	1.732674	0.954592
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000	2.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000	2.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000	4.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

از کمترین و بیشترین مقادیر داده ها میتوان نتیجه گرفت داده های پرت نداریم، همچنین داده های خالی یا صفر نداریم.

در این مسله وجود داده ی تکرار موجب خطا نخواهد شد لذا از بررسی این مورد صرف نظر شده است.

در مرحله بعد با استفاده از کد Replace بجای ارقام ۲ و ۴ در ستون Class اعداد ۰ و ۱ را جایگزین میکنیم.

```
dataset['Class']= dataset['Class'].replace(2,0)
dataset['Class']= dataset['Class'].replace(4,1)
```

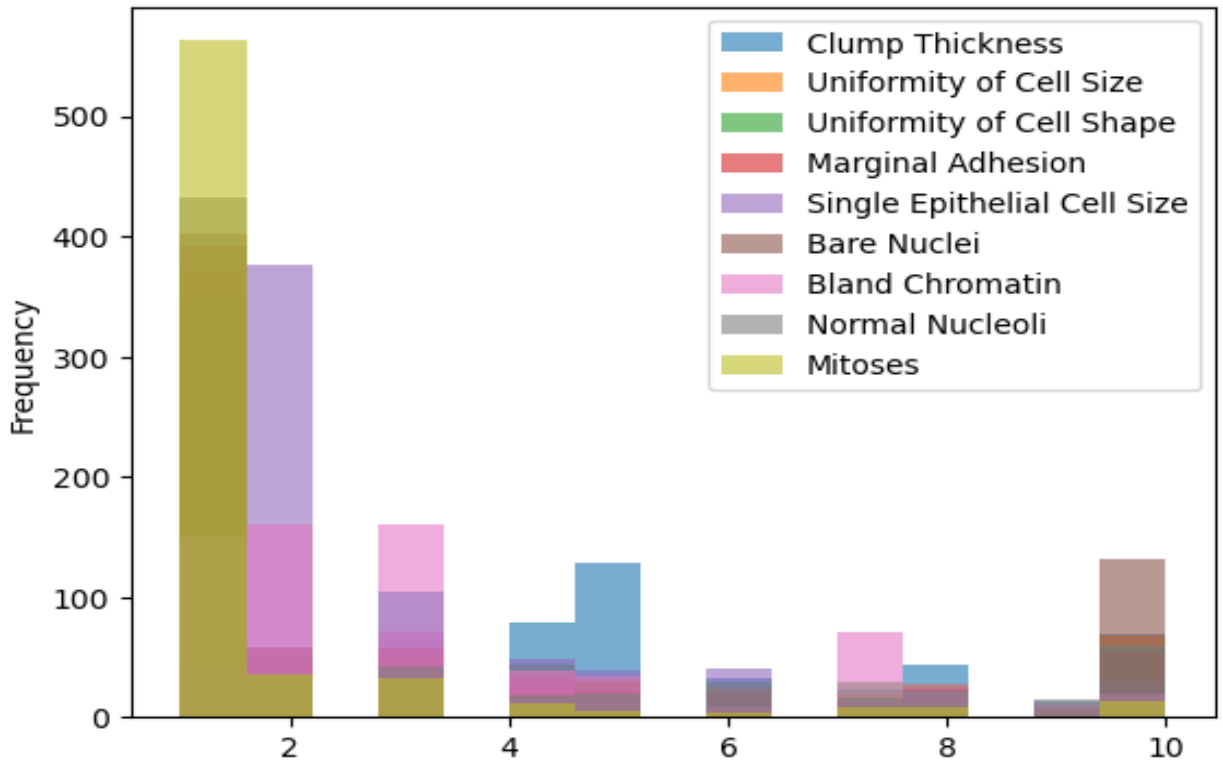
از آنجایی که مقادیر داده شده برای ویژگی های ۹ گانه همگی در بازه ۱ تا ۱۰ هستند نیازی به نرمال سازی داده ها نخواهد بود.

ب) توزیع دادگان را برای تمام ۹ ویژگی بدست آورید. داده‌های با نوع تومور مختلف را با رنگ های متفاوت از یکدیگر نمایش دهید. پراکندگی داده‌ها را بر حسب ویژگی‌های مختلف نشان دهید. بر پایهٔ این پراکندگی‌ها برداشت خود را از ماهیت داده‌ها بیان کنید (راهنمایی: برای نمایش توزیع داده‌ها میتوانید از نمودارهای نقشه گرمایی، گسسته، هیستوگرام یا جعبه ای استفاده کنید.)

پاسخ:

برای نمایش توزیع دادگان با استفاده از تابع Plot یک هیستوگرام با پارامتر های نشان داده شده در تصویر ذیل رسم میکنیم.

```
data_distribution1=dataset.drop('Class',axis=1).copy()
data_distribution1.plot(kind='hist',bins=15,alpha=.6)
plt.show()
```



همه ی ۹ ویژگی با رنگ مختص به خود در هیستوگرام مشخص شده اند و با تنظیم پارامتر آلفا شفافیت رنگ تنظیم شده است تا دادگانی که تداخل پیدا کرده اند مشخص باشند، همچنین با پارامتر bins میزان بزرگی میله های هیستوگرام تنظیم شده است.

در مرحله بعد برای نمایش داده های با نوع تومور مختلف از یک حلقه if مشخصات جدول یعنی رنگ بندی وابسته به مقدار هر سطر در ستون Class سطر هارا به دو رنگ سبز و قرمز تقسیم میکنیم:

در حلقه if تعریف شده است که تک تک سطر هارا بررسی کرده و با توجه به مقدار هر سطر در ستون Class از بین ۰ و ۱ رنگ بندی آن سطر را سبز یا قرمز قرار دهد.

```
def color_row(row):
    if row['Class'] == 0:
        return ['background-color: green'] * len(row)
    elif row['Class'] == 1:
        return ['background-color: red'] * len(row)
    else:
        return [''] * len(row)
table = dataset.style.apply(color_row, axis=1)
table
```

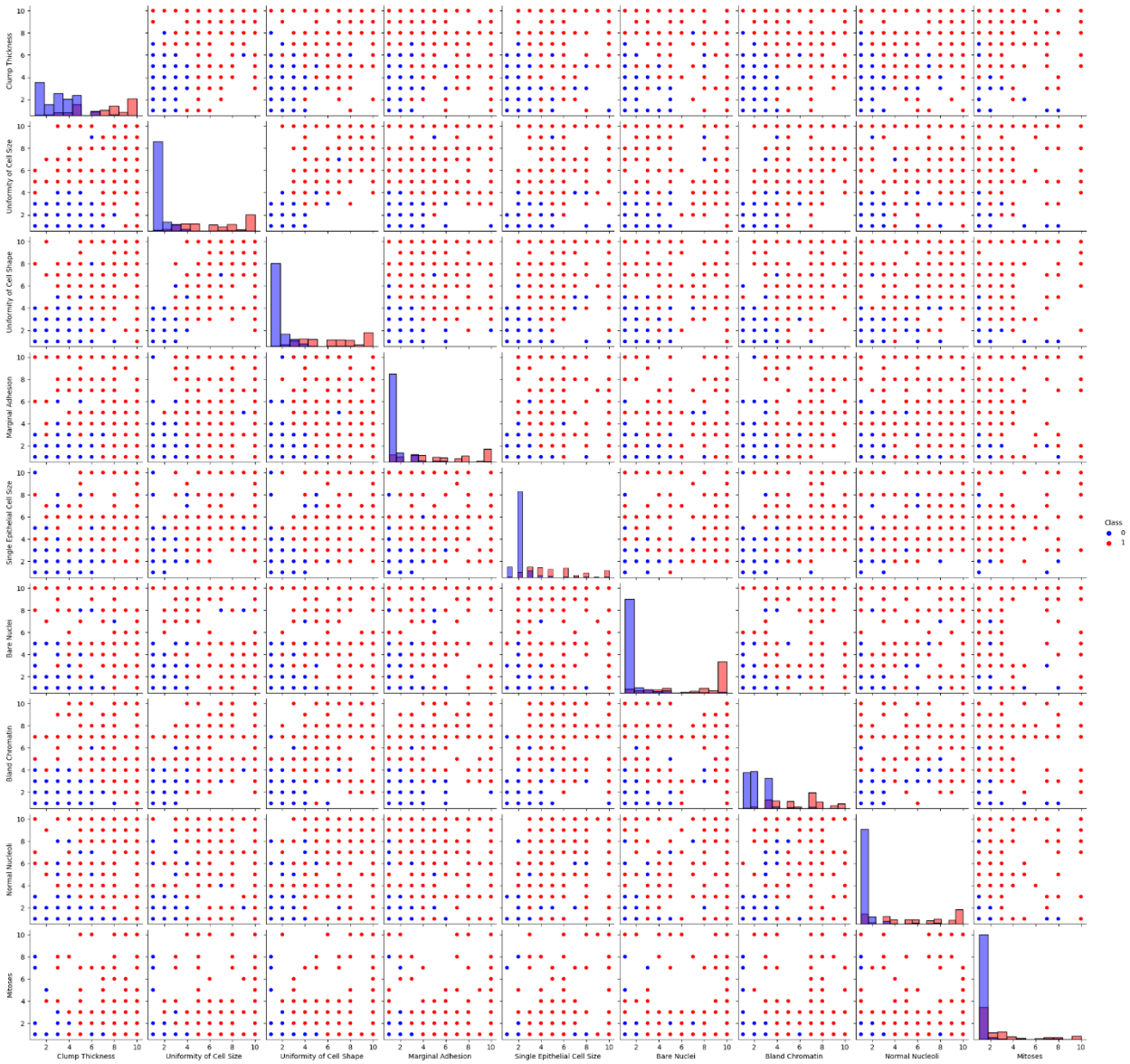
119	4	2	1	1	2	2	3	1	1	0
120	10	10	10	2	10	10	5	3	3	1
121	5	3	5	1	8	10	5	3	1	1
122	5	4	6	7	9	7	8	10	1	1
123	1	1	1	1	2	1	2	1	1	0
124	7	5	3	7	4	10	7	5	5	1
125	3	1	1	1	2	1	3	1	1	0
126	8	3	5	4	5	10	1	6	2	1
127	1	1	1	1	10	1	1	1	1	0
128	5	1	3	1	2	1	2	1	1	0
129	2	1	1	1	2	1	3	1	1	0
130	5	10	8	10	8	10	3	6	3	1
131	3	1	1	1	2	1	2	2	1	0

پراکندگی داده ها بر حسب ویژگی های مختلف:

با استفاده از sns.pairplot از seaborn یک نمودار از پراکندگی داده ها با رنگ بندی برحسب نوع تومور رسم میکنیم:

```
colors = {0: 'blue', 1: 'red'}
sns.pairplot(dataset, hue='Class', palette=colors, diag_kind='hist')
```

پس از تعریف رنگ ها برای هر نوع Class انرا در pairplot در پارامتر palette استفاده میکنیم و نوع نمودار را از نوع هیستوگرامی و مبنای تقسیم بندی را ستون Class معرفی میکنیم.



بر پایه این خروجی ها میتوان به تجمع اکثریت داده ها در بازه ۰ تا ۲ پی برد که این موضوع میتواند باعث تمرکز مدل روی داده های این مقادیر بشود(این مسله را از خروجی تابع describe در قسمت قبلی سوال هم میشد دریافت کرد) و برای مقادیر دیگر دچار خطا

شود، همینطور میتوان به این موضوع پی برد که هرقدر مقدار ویژگی ها بیشتر باشد داده از نوع کلاس ۱ و تومور بد خیم خواهد بود.

ج) رابطه و تاثیرگذاری هر کدام از پارامترها بر نوع تومور را پیدا کنید. (راهنمایی: می‌توانید از معیارهای آماری مانند کوریلیشن استفاده کنید تا میزان تاثیر داده‌ها بر روی خروجی و حتی رابطهٔ آن‌ها با یکدیگر را مشاهده کنید. برای نمایش هم می‌توانید از pair-plot در کتابخانه seaborn استفاده کنید.)

پاسخ:

```
corr1 = dataset['Clump Thickness'].corr(dataset['Class'])
corr2 = dataset['Uniformity of Cell Size'].corr(dataset['Class'])
corr3 = dataset['Uniformity of Cell Shape'].corr(dataset['Class'])
corr4 = dataset['Marginal Adhesion'].corr(dataset['Class'])
corr5 = dataset['Single Epithelial Cell Size'].corr(dataset['Class'])
corr6 = dataset['Bare Nuclei'].corr(dataset['Class'])
corr7 = dataset['Bland Chromatin'].corr(dataset['Class'])
corr8 = dataset['Normal Nucleoli'].corr(dataset['Class'])
corr9 = dataset['Mitoses'].corr(dataset['Class'])
print("Corrolation between Clump Thickness and Cancer type  : {:.3f}".format(corr1))
print("Corrolation between Uniformity of Cell Size and Cancer type  : {:.3f}".format(corr2))
print("Corrolation between Uniformity of Cell Shape and Cancer type  : {:.3f}".format(corr3))
print("Corrolation between Marginal Adhesion and Cancer type  : {:.3f}".format(corr4))
print("Corrolation between Single Epithelial Cell Size and Cancer type  : {:.3f}".format(corr5))
print("Corrolation between Bare Nuclei and Cancer type  : {:.3f}".format(corr6))
print("Corrolation between Bland Chromatin and Cancer type  : {:.3f}".format(corr7))
print("Corrolation between Normal Nucleoli and Cancer type  : {:.3f}".format(corr8))
print("Corrolation between Mitoses and Cancer type  : {:.3f}".format(corr9))
```

برای بدست آوردن رابطه و تاثیر پارامتر ها بر نوع تومور از دو تابع sns.pairplot و corr استفاده شده است .

```
Corrolation between Clump Thickness and Cancer type  : 0.715
Corrolation between Uniformity of Cell Size and Cancer type  : 0.821
Corrolation between Uniformity of Cell Shape and Cancer type  : 0.822
Corrolation between Marginal Adhesion and Cancer type  : 0.706
Corrolation between Single Epithelial Cell Size and Cancer type  : 0.691
Corrolation between Bare Nuclei and Cancer type  : 0.823
Corrolation between Bland Chromatin and Cancer type  : 0.758
Corrolation between Normal Nucleoli and Cancer type  : 0.719
Corrolation between Mitoses and Cancer type  : 0.423
```

نتیجه:

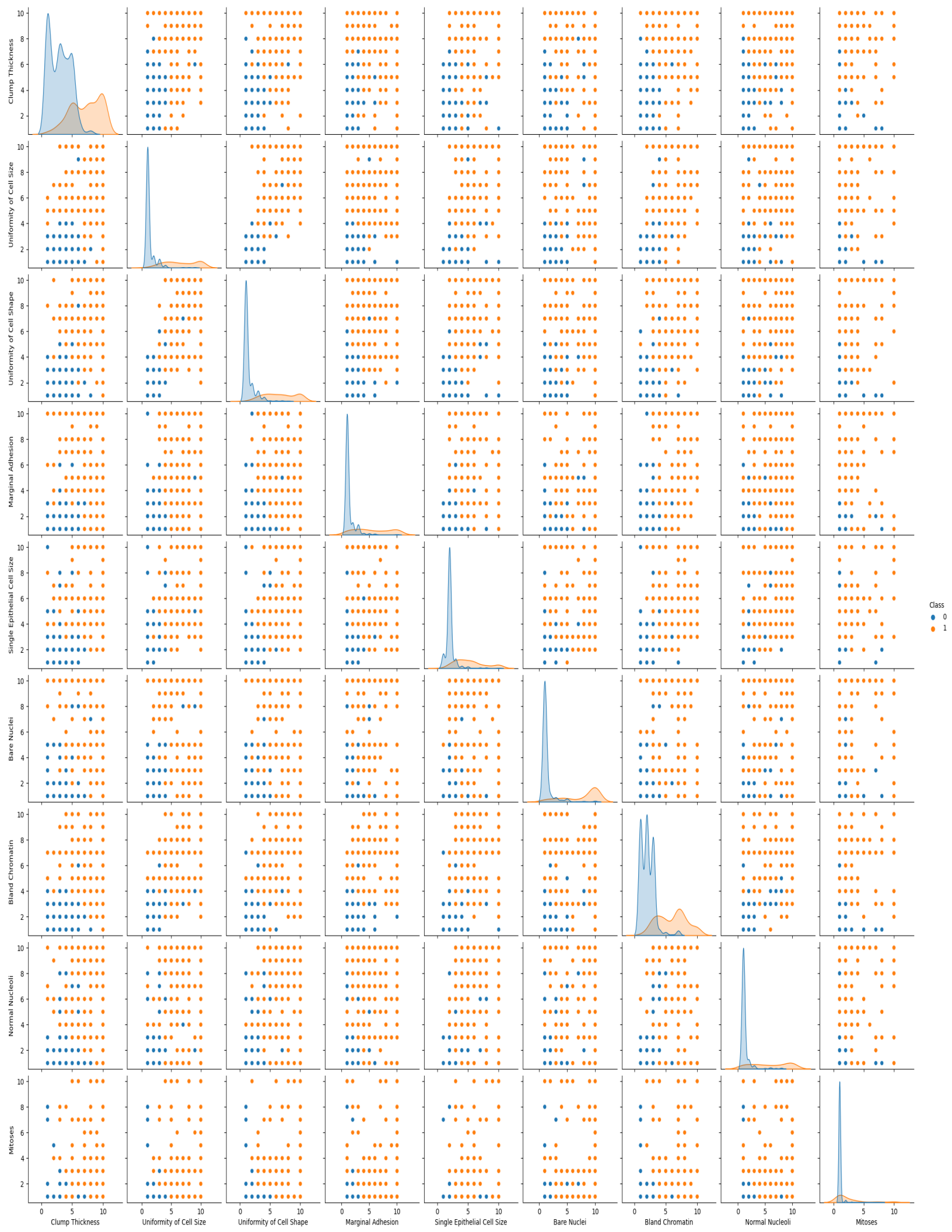
در صفحه بعد رابطه نموداری بین ویژگی ها و نوع سرطان نمایش داده شده است.

مقدار corr از -۱ تا +۱ متغیر خواهد بود و مقادیری که به یک نزدیکتر باشند بیشترین تاثیر گذاری را دارند. نتیجه:

ویژگی های با بیشترین تاثیر بر نوع تومور به ترتیب:

Bare Nuclei ، Uniformity of cell shape و Uniformity of cell size.

ویژگی با کمترین تاثیر بر نوع تومور: Mitoses با تاثیر گذاری حدود نصف Bare Nuclei


Clump Thickness
Uniformity of Cell Size
Uniformity of Cell Shape
Marginal Adhesion
Single Epithelial Cell Size
Bare Nuclei
Bland Chromatin
Normal Nucleoli
Mitoses
Class
0
1

د) داده ها را به سه بخش آموزش (training)، ارزیابی (validation) و آزمایش (test) تقسیم کنید. پیشنهاد می‌شود ۸۰٪ کل داده‌ها به آموزش، ۱۰٪ به ارزیابی و ۱۰٪ به آزمایش اختصاص داده شود. در گام بعد با استفاده از الگوریتم رگرسیون لجستیک، مدلی را برای پیش‌بینی خروجی تربیت کنید و سپس با استفاده از روش k-fold cross validation (با k=5) بهترین عملکرد مدل را بدست آورید (این روش در ادامهٔ درس معرفی خواهد شد.). (توضیح بیشتر: در یادگیری ماشین هر مدل برای تنظیم پارامترهای

پاسخ:

ابتدا داده هارا به دو بخش ویژگی ها و نوع سرطان تقسیم میکنیم.

```
y=dataset['Class']
x=dataset.drop(['Class'],axis=1)
```

سپس با استفاده از تابع train_test_split از کتابخانه sklearn داده ها را به سه بخش با درصد های ۸۰ درصد برای تمرین ، ۱۰ درصد برای ارزیابی و ۱۰ درصد برای آزمون تقسیم میکنیم.

در این مرحله جهت تکرار پذیری نتایج بدست امده یک رندم استیت تعریف میشود، در واقع الگوی انتخاب داده ها ذخیره شده است در رندم استیت ۱، بدیهی است با رندم استیت های دیگر الگو تغییر کند و پاسخ ها متفاوت شود.

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.2,random_state=1)
x_test, x_val, y_test, y_val =train_test_split(x_test,y_test, test_size= 0.5,random_state=1)
```

سپس یک مدل رگرسیون لاجستیک از کتابخانه sklearn.linear فراخوانی میکنیم و با داده های آموزشی(تمرین) مدل را تربیت میکنیم. همینطور پاسخ مدل به داده های ازمایش را جهت مقایسه با پاسخ صحیح انها در ماتریس سردرگمی استخراج میکنیم.

```
reg=LogisticRegression()
reg.fit(x_train,y_train)
y_predict = reg.predict(x_test)
```

در مرحله بعد با توجه به خواسته سوال بهترین عملکرد مدل را با استفاده از روش k fold validation بدست میاوریم.

برای این کار تابع kFold را از کتابخوانه skLearn فراخوانی میکنیم و با مقدار k=5 بهترین عملکرد تابع را بدست میاوریم.

```
k=5
kf=KFold(n_splits=k,shuffle=True,random_state=1)
results=cross_val_score(reg,x,y,cv=kf)
scoring='accuracy'
mean_accuracy=results.mean()
best_accuracy=results.max()
print(f"Measured accuracies:{results}")
print(f"Mean accuracy on k fold by k=5 validation set:{mean_accuracy*100:.3f}")
print(f"Best accuracy on k fold by k=5 validation set:{best_accuracy*100:.3f}")
```

در خط اول مقدار ۵ را به k اختصاص میدهیم، تابع kFold با پارامتر های رندم استیت برای ذخیره الگوی انتخاب داده ها و تکرار پذیری داده و عدم تغییر در الگوی انتخاب داده ها توسط تابع در هر بار ران کردن برنامه.

n_splits تعداد تقسیمات داده ها که همان مقدار k است و تنظیم انتخاب بدون ترتیب داده ها با پارامتر shuffle تنظیم شده است.

با استفاده از تابع cross_val_score دقت هر مرحله را ذخیره میکنیم،پارامتر های این تابع : مدل رگرسیون، داده های ورودی به تفکیک ویژگی ها و کلاس ها و پارامتر cv که تعیین کننده الگوی تقسیم بندی داده ها است.

سپس بهترین دقت و میانگین همه ی دقت و همینطور همه ی دقت ها چاپ میشوند.

```
Measured accuracies:[0.98540146 0.94890511 0.96350365 0.94852941 0.97794118]
Mean accuracy on k fold by k=5 validation set:96.486
Best accuracy on k fold by k=5 validation set:98.540
```

بنابراین بهترین دقت مدل ۹۸.۵۴ است.

ه) بر روی داده‌های آزمایش، ماتریس سردرگمی را تشکیل دهید (این ماتریس نیز در ادامهٔ درس معرفی خواهد شد) و نتایج را تحلیل کنید. میزان دقت بدست آمده را مناسب می‌دانید یا خیر؟ پیشنهادهای خود را برای افزایش دقت ارائه دهید.

پاسخ:

برای محاسبه ماتریس سردرگمی تابع confusion_matrix از کتابخانه sklearn را بکار میگیریم.

```
con_mat=confusion_matrix(y_test,y_predict)
print(con_mat)
```

در این ماتریس پاسخ داده های ازمایش و پاسخ های پیش بینی شده توسط مدل برای داده های تست به عنوان پارامتر ها وارد شدند.

```
[[44  0]
 [ 2 22]]
```

طبق این ماتریس نتایج برای ۶۶ داده از ۶۸ داده بدرستی توسط مدل پیش بینی شده اند و نتایج برای ۲ داده هم به اشتباه پیش بینی شده اند که دقت ۹۷.۰۵ را نشان میدهد.

تعداد داده هایی که بدرستی بد خیم تشخیص داده شده اند: ۴۴

تعداد داده هایی که به اشتباه بد خیم تشخیص داده شده اند: ۰

تعداد داده هایی که بدرستی خوش خیم تشخیص داده شده اند: ۲۲

تعداد داده هایی که به اشتباه خوش خیم تشخیص داده شده اند: ۲

برای بدست اوردن مقادیر دقت، صحت، حساسیت(به یاد آوری) و نرخ خطا از تابع زیر از کتابخانه sklearn استفاده کردیم.

پارامتر های این تابع همان پارامتر های ماتریس سردرگمی هستند.

```
print(classification_report(y_test, y_predict))
```

نتیجه:

	precision	recall	f1-score
0	0.96	1.00	0.98
1	1.00	0.92	0.96
accuracy			0.97

دقت محاسبه مدل برای تومور های بدخیم صد در صد است و برای تومور های خوش خیم ۹۶ درصد است که دقت بسیار خوبی است، برای افزایش دقت مدل میتوان داده های ورودی را افزایش داد یا از مدل های رگرسیون دیگری استفاده کرد.

و) **امتیازی**: بعد از آموزش، مواردی که به غلط توسط مدل پیش‌بینی شده را جدا کنید. با استفاده از تحلیل آماری یا تفسیر بصری علت این پیش‌بینی غلط را توضیح دهید و با ذکر دلیل، در جهت بهبود دقت مدل تلاش کنید.

با استفاده از تابع np.where از کتابخانه numpy داده هایی که به اشتباه پیش بینی شده اند را استخراج میکنیم، در این تابع پارامتری برای توضیح وجود ندارد.

```
incorrect_predictions=np.where(y_predict!=y_test)[0]
print('Incorrently predicted samples row:',incorrect_predictions)
```

نتیجه:

```
Incorrently predicted samples row: [31 41]
```

سطر های ۳۱ و ۴۱:

Clump Thickness	5
Uniformity of Cell Size	6
Uniformity of Cell Shape	5
Marginal Adhesion	6
Single Epithelial Cell Size	10
Bare Nuclei	1
Bland Chromatin	3
Normal Nucleoli	1
Mitoses	1
Class	1

Name: 41, dtype: int64

Clump Thickness	10
Uniformity of Cell Size	7
Uniformity of Cell Shape	7
Marginal Adhesion	3
Single Epithelial Cell Size	8
Bare Nuclei	5
Bland Chromatin	7
Normal Nucleoli	4
Mitoses	3
Class	1

Name: 31, dtype: int64

این خطا بدلیل خاص بودن مقادیر داده است، در این داده ها ویژگی هایی که بیشترین تاثیرگذاری را دارند در سطح بالایی هستند و/یا ویژگی هایی که تاثیرگذاری خیلی کمتری دارند مقادیرشون بسیار پایین است(data_31)، همینطور طبق پیش بینی قبلی انجام شده انتظار میرفت مدل برای داده هایی که مقادیر ویژگی بالایی دارند دچار خطا شود(data_41).

بخش دوم: پیش‌بینی عمر مفید مواد دی‌الکتریک

مواد دی‌الکتریک به دلیل توانایی ذخیره‌ی بار الکتریکی (مانند عملکرد خازن‌ها) بصورت گسترده در صنعت مورد استفاده قرار می‌گیرند. از جمله بررسی‌هایی که در مورد این مواد صورت می‌گیرد، تعیین حداکثر ولتاژ قابل اعمال به آن‌ها در دمای کاری مشخص و برای عمر مفید مشخص است.

دادگانی که در این بخش مورد استفاده قرار می‌گیرد (فایل Performance-Degradation Data Nelson.xlsx) شامل نتایج ۱۲۸ آزمایش تعیین ولتاژ بیشینه است. هر نمونه شامل دو ویژگیِ عمر مفید (بر حسب هفته) و دمای کاری (بر حسب درجه سلسیوس) در ستون‌های اول و دوم و یک خروجیِ ولتاژ بیشینهٔ مجاز دی‌الکتریک (بر حسب کیلوولت) در ستون سوم است.

الف) به کمک نرم افزار پایتون و کتابخانه‌های مناسب، داده‌های فایل را وارد و به کمک دستور مناسب از کتابخانه‌ی sklearn، رگرسیون را با کرنل‌های خطی، RBF، چندجمله‌ای درجه ۲ و سیگموئیدی انجام داده و خطای مطلق میانگین (mean absolute error) و امتیاز R2 (R2-score) را در هر کدام به کمک روش k-fold cross validation (با k=4 یعنی در هر حالت ٪۲۵ از داده‌ها با انتخاب تصادفی پیش فرض sklearn برای آزمایش در نظر گرفته شود) به دست آورده و عملکرد چهار تابع کرنل (میانگین امتیاز به دست آمده برای داده‌های آزمایش) را مقایسه نمایید.

پاسخ:

وارد کردن کتابخانه های مورد استفاده و داده ها:

```
import pandas as pd
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
dataset = pd.read_excel(r'C:\Users\Nima\Desktop\AI Project 2\Qustion2\Performance-Degradation Data Nelson.xlsx')
```

با استفاده از تابع describe مقادیر موجود در داده ها را مورد مطالعه قرار میدهیم، ویژگی ها نیاز به نرمال سازی دارند و پرتی و مقدار صفر دیده نمیشود.

```
dataset.describe()
```

	y	x1	x2
count	128.000000	128.000000	128.000000
mean	11.238047	21.875000	232.500000
std	4.166401	22.270704	35.227048
min	1.000000	1.000000	180.000000
25%	10.000000	3.500000	213.750000
50%	12.000000	12.000000	237.500000
75%	13.525000	36.000000	256.250000
max	18.500000	64.000000	275.000000

از آنجا که مقادیر ویژگی ها مختلف است لذا نیاز است داده ها نرمال سازی شوند لذا:

```
dataset['x1'] = (dataset['x1'] - dataset['x1'].min()) / (dataset['x1'].max() - dataset['x1'].min()) * 0.9 + 0.1
dataset['x2'] = (dataset['x2'] - dataset['x2'].min()) / (dataset['x2'].max() - dataset['x2'].min()) * 0.9 + 0.1
dataset['y'] = (dataset['y'] - dataset['y'].min()) / (dataset['y'].max() - dataset['y'].min()) * 0.9 + 0.1
```

نرمال سازی داده ها به گونه ای بوده که کمترین مقدار ۰.۱ و بیشترین مقدار ۱ باشد.(نرمال سازی بین ۰.۱ تا ۱)

همچنین بعد از نرمال سازی با تابع Round مقدار قسمت اعشاری داده ها تا سه رقم اعشار گرد شدند.

سپس بعد از تعریف مقدار k تابع k Fold از کتابخوانه sklearn فراخوانی میشود.

پارامترهای این تابع: رندم استیت برای ذخیره روش انتخاب داده ها و تکرار پذیری داده و عدم تغییر در الگوی انتخاب داده ها توسط تابع در هر بار اجرا کردن برنامه.

n_splits تعداد تقسیمات داده ها که همان مقدار k است و تنظیم انتخاب بدون ترتیب داده ها با پارامتر shuffle تنظیم شده است.

```
k = 4
kf = KFold(n_splits=k, shuffle=True, random_state=1)
```

در ادامه با فرخوانی تابع SVR از کتابخانه sklearn رگرسیون هارا با کرنل های خطی،RBF،چند جمله ای درجه ۲ و سیگموید و خطای مطلق میانگین و امتیاز R2 را به کمک تابع k fold variation بدست میاوریم.

کرنل خطی:

```
#Linear_Kernel
Kernel_linear = SVR(kernel='linear')
LK_mae_scores = []
LK_r2_scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    Kernel_linear.fit(x_train, y_train)
    LK_y_pred = Kernel_linear.predict(x_test)
    LK_mae = mean_absolute_error(y_test, LK_y_pred)
    LK_mae_scores.append(LK_mae)
    LK_r2 = r2_score(y_test, LK_y_pred)
    LK_r2_scores.append(LK_r2)
avg_LK_mae = sum(LK_mae_scores) / k
avg_LK_r2 = sum(LK_r2_scores) / k
print(f'  linear kernel:  Average Mean Absolute Error: {avg_LK_mae:.2f}')
print(f'                  Average R2 Score: {avg_LK_r2:.2f}')
```

طبق تصویر نوع کرنل مورد استفاده در تابع SVR را خطی قرار میدهیم،سپس دو لیست خالی جهت ذخیره داده های خروجی از حلقه for میسازیم.

در حلقه for پس از تقسیم داده ها به دو بخش آموزش و آزمون رگرسیون را بر روی داده ها آموزش میدهیم، سپس مقادیر خروجی رگرسیون را در LK_y_pred که مخفف پاسخ کرنل خطی است ذخیره میکنیم، مدل را روی داده های آموزشی فیت میکنیم و جهت ارزیابی مدل، پاسخ مدل روی داده های آزمایش را با پاسخ صحیح شان مقایسه میکنیم.

برای این منظور طبق خواست سوال خطای مطلق میانگین و امتیاز R2 محاسبه میشوند و با تابع append به لیست ایجاد شده در مرحله قبل اضافه میشوند.

سپس میانگین هردو مورد محاسبه و چاپ میشوند.

کرنل RBF:

```
#RBF_Kernel
Kernel_RBF = SVR(kernel='rbf', C=1.0, gamma='scale', epsilon=0.1)
RK_mae_scores = []
RK_r2_scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    Kernel_RBF.fit(x_train, y_train)
    RK_y_pred = Kernel_RBF.predict(x_test)
    RK_mae = mean_absolute_error(y_test, RK_y_pred)
    RK_mae_scores.append(RK_mae)
    RK_r2 = r2_score(y_test, RK_y_pred)
    RK_r2_scores.append(RK_r2)
avg_RK_mae = sum(RK_mae_scores) / k
avg_RK_r2 = sum(RK_r2_scores) / k
print(f'  RBF kernel:  Average Mean Absolute Error: {avg_RK_mae:.2f}')
print(f'               Average R2 Score: {avg_RK_r2:.2f}')
```

طبق تصویر نوع کرنل مورد استفاده در تابع SVR را RBF قرار میدهیم،سپس دو لیست خالی جهت ذخیره داده های خروجی از حلقه for میسازیم.

در حلقه for پس از تقسیم داده ها به دو بخش آموزش و آزمون رگرسیون را بر روی داده ها آموزش میدهیم، سپس مقادیر خروجی رگرسیون را در RK_y_pred که مخفف پاسخ کرنل RBF است ذخیره میکنیم، مدل را روی داده های آموزشی فیت میکنیم و جهت ارزیابی مدل، پاسخ مدل روی داده های آزمایش را با پاسخ صحیح شان مقایسه میکنیم.

برای این منظور طبق خواست سوال خطای مطلق میانگین و امتیاز R2 محاسبه میشوند و با تابع append به لیست ایجاد شده در مرحله قبل اضافه میشوند.

سپس میانگین هردو مورد محاسبه و چاپ میشوند.

این تابع کرنل دارای سه پارامتر است:

C: برابر با ۱ گرفته شده است تا حساسیت مدل به داده های نویزی کم باشد.

Gamma: scale در نظر گرفته شده است تا بر اساس واریانس داده ها تنظیم شود.

Epsilon: این مقدار برابر ۰.۱ در نظر گرفته شده است تا حاشیه مدل برای داده های آموزش مناسب باشد.

کرنل چند جمله ای درجه ۲:

```
#Quadratic_polynomial_Kernel
Kernel_QuadPoly= SVR(kernel='poly', degree=2)
QPK_mae_scores = []
QPK_r2_scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    Kernel_QuadPoly.fit(x_train, y_train)
    QPK_y_pred = Kernel_QuadPoly.predict(x_test)
    QPK_mae = mean_absolute_error(y_test, QPK_y_pred)
    QPK_mae_scores.append(QPK_mae)
    QPK_r2 = r2_score(y_test, QPK_y_pred)
    QPK_r2_scores.append(QPK_r2)
avg_QPK_mae = sum( QPK_mae_scores) / k
avg_QPK_r2 = sum( QPK_r2_scores) / k
print(f'  Quadratic polynomial kernel:  Average Mean Absolute Error: {avg_QPK_mae:.2f}')
print(f'                                Average R2 Score: {avg_QPK_r2:.2f}')
```

طبق تصویر نوع کرنل مورد استفاده در تابع SVR را چند جمله ای قرار میدهیم،سپس دو لیست خالی جهت ذخیره داده های خروجی از حلقه for میسازیم.

در حلقه for پس از تقسیم داده ها به دو بخش آموزش و آزمون رگرسیون را بر روی داده ها آموزش میدهیم، سپس مقادیر خروجی رگرسیون را در QPK_y_pred که مخفف پاسخ کرنل Quadratic_polynomial است ذخیره میکنیم، مدل را روی داده های آموزشی فیت میکنیم و جهت ارزیابی مدل، پاسخ مدل روی داده های آزمایش را با پاسخ صحیح شان مقایسه میکنیم.

برای این منظور طبق خواست سوال خطای مطلق میانگین و امتیاز R2 محاسبه میشوند و با تابع append به لیست ایجاد شده در مرحله قبل اضافه میشوند.

سپس میانگین هردو مورد محاسبه و چاپ میشوند.

پارامتر تعریف شده درجه تابع چند جمله ای است.

کرنل سیگموییید:

```
#Sigmoid_Kernel
Kernel_Sigmoid= SVR(kernel='sigmoid' ,C=.001)
SK_mae_scores = []
SK_r2_scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    Kernel_Sigmoid.fit(x_train, y_train)
    SK_y_pred = Kernel_Sigmoid.predict(x_test)
    SK_mae = mean_absolute_error(y_test, SK_y_pred)
    SK_mae_scores.append(SK_mae)
    SK_r2 = r2_score(y_test, SK_y_pred)
    SK_r2_scores.append(SK_r2)
avg_SK_mae = sum(SK_mae_scores) / k
avg_SK_r2 = sum(SK_r2_scores) / k
print(f'  Sigmoid kernel:  Average Mean Absolute Error: {avg_SK_mae:.2f}')
print(f'                   Average R2 Score: {avg_SK_r2:.2f}')
```

طبق تصویر نوع کرنل مورد استفاده در تابع SVR را سیگموییید قرار میدهیم،سپس دو لیست خالی جهت ذخیره داده های خروجی از حلقه for میسازیم.(مقدار پارامتر C تعریف شده در اینجا نتیجه چندین بار اجرا برنامه با مقادیر مختلف امتیازات است)

در حلقه for پس از تقسیم داده ها به دو بخش آموزش و آزمون رگرسیون را بر روی داده ها آموزش میدهیم، سپس مقادیر خروجی رگرسیون را در SK_y_pred که مخفف پاسخ کرنل sigmoid است ذخیره میکنیم، مدل را روی داده های آموزشی فیت میکنیم و جهت ارزیابی مدل، پاسخ مدل روی داده های آزمایش را با پاسخ صحیح شان مقایسه میکنیم.

برای این منظور طبق خواست سوال خطای مطلق میانگین و امتیاز R2 محاسبه میشوند و با تابع append به لیست ایجاد شده در مرحله قبل اضافه میشوند.

سپس میانگین هردو مورد محاسبه و چاپ میشوند.

نتیجه:

```
linear kernel:  Average Mean Absolute Error: 0.10
                Average R2 Score: 0.68

RBF kernel:  Average Mean Absolute Error: 0.07
             Average R2 Score: 0.85

Quadratic polynomial kernel:  Average Mean Absolute Error: 0.08
                              Average R2 Score: 0.81

Sigmoid kernel:  Average Mean Absolute Error: 0.16
                 Average R2 Score: -0.04
```

نتایج نشان میدهد بهترین کرنل مربوط به چند جمله ای درجه ۲ بوده است و کرنل سیگموییید اصلا مناسب عمل نمیکند.

ب) خواسته‌ی مورد الف را به کمک L2-regularization با پارامتر alpha=1, 2 (دو مقدار) برای توابع کرنل بخش الف (چهار تابع) به دست آورده و نتایج را مقایسه نمایید. با توجه به تغییرات به وجود آمده در دقت رگرسیون درداده‌های آزمایش نسبت به بخش قبل، چه نتایجی می‌توان گرفت؟

پاسخ:

کرنل خطی:

با فراخوانی تابع Ridge از کتابخانه sklearn.linear انرا با آلفا ۱ بر روی داده های آموزشی فیت میکنیم.

```
#Linear_kernel
L2_regularization_Linear_1 = Ridge(alpha=1,)
L2_regularization_Linear_1.fit(x_train, y_train)
Lin1_y_pred = L2_regularization_Linear_1.predict(x_test)
Lin1_mae = mean_absolute_error(y_test, Lin1_y_pred)
Lin1_r2 = r2_score(y_test, Lin1_y_pred)
print(f'  Linear kernel & alpha:1  Mean Absolute Error: {Lin1_mae:.2f}')
print(f'                           R2 Score: {Lin1_r2:.2f}')
print()
L2_regularization_Linear_2 = Ridge(alpha=2,)
L2_regularization_Linear_2.fit(x_train, y_train)
Lin2_y_pred = L2_regularization_Linear_2.predict(x_test)
Lin2_mae = mean_absolute_error(y_test, Lin2_y_pred)
Lin2_r2 = r2_score(y_test, Lin2_y_pred)
print(f'  Linear kernel & alpha:2  Mean Absolute Error: {Lin2_mae:.2f}')
print(f'                           R2 Score: {Lin2_r2:.2f}')
print()
```

سپس مقادیر پیش بینی شده توسط مدل را ذخیره و خطای میانگین مطلق و امتیاز R2 را محاسبه میکنیم وسپس چاپ میکنیم.

همین روند را برای مقدار آلفا ۲ تکرار میکنیم.

کرنل RBF:

تابع ridge فقط برای کرنل خطی قابل استفاده است. پس

با فراخوانی تابع kernelRidge از کتابخانه sklearn.ridge و قرار دادن نوع کرنل rbf با آلفا ۱ انرا بر روی داده های آموزشی فیت میکنیم.

```
#RBF_kernel
L2_regularization_Rbf_1 = KernelRidge(kernel='rbf', alpha=1.0)
L2_regularization_Rbf_1.fit(x_train, y_train)
RBF1_y_pred = L2_regularization_Rbf_1.predict(x_test)
RBF1_mae = mean_absolute_error(y_test, RBF1_y_pred)
RBF1_r2 = r2_score(y_test, RBF1_y_pred)
print(f'  PBF kernel & alpha:1  Mean Absolute Error: {RBF1_mae:.2f}')
print(f'                        R2 Score: {RBF1_r2:.2f}')
print()
L2_regularization_Rbf_2 = KernelRidge(kernel='rbf', alpha=2.0)
L2_regularization_Rbf_2.fit(x_train, y_train)
RBF2_y_pred = L2_regularization_Rbf_2.predict(x_test)
RBF2_mae = mean_absolute_error(y_test, RBF2_y_pred)
RBF2_r2 = r2_score(y_test, RBF2_y_pred)
print(f'  PBF kernel & alpha:2  Mean Absolute Error: {RBF2_mae:.2f}')
print(f'                        R2 Score: {RBF2_r2:.2f}')
```

سپس مقادیر پیش بینی شده توسط مدل را ذخیره و خطای میانگین مطلق و امتیاز R2 را محاسبه میکنیم وسپس چاپ میکنیم.

همین روند را برای مقدار آلفا ۲ تکرار میکنیم.

کرنل چند جمله ای درجه ۲: با فراخوانی تابع kernelRidge از کتابخانه sklearn.ridge و نوع کرنل چند جمله ای با آلفا ۱ انرا بر روی داده های آموزشی فیت میکنیم.

```
#Quadratic_polynomial_Kernel
L2_regularization_QP_1 = KernelRidge(kernel='polynomial', degree=2, alpha=1)
L2_regularization_QP_1.fit(x, y)
QP1_y_pred = L2_regularization_QP_1.predict(x)
QP1_mae = mean_absolute_error(y, QP1_y_pred)
QP1_r2 = r2_score(y, QP1_y_pred)
print(f'  Quadratic_polynomial_Kernel & alpha:1  Mean Absolute Error: {QP1_mae:.2f}')
print(f'                                         R2 Score: {QP1_r2:.2f}')
print()
L2_regularization_QP_2 = KernelRidge(kernel='polynomial', degree=2, alpha=2)
L2_regularization_QP_2.fit(x, y)
QP2_y_pred = L2_regularization_QP_2.predict(x)
QP2_mae = mean_absolute_error(y, QP2_y_pred)
QP2_r2 = r2_score(y, QP2_y_pred)
print(f'  Quadratic_polynomial_Kernel & alpha:2  Mean Absolute Error: {QP2_mae:.2f}')
print(f'                                         R2 Score: {QP2_r2:.2f}')
```

سپس مقادیر پیش بینی شده توسط مدل را ذخیره و خطای میانگین مطلق و امتیاز R2 را محاسبه میکنیم وسپس چاپ میکنیم.

همین روند را برای مقدار آلفا ۲ تکرار میکنیم.

کرنل سیگموید:

با فراخوانی تابع kernelRidge از کتابخانه sklearn.ridge و قرار دادن نوع کرنل sigmoid با آلفا ۱ انرا بر روی داده های آموزشی فیت میکنیم.

```
#Sigmoid_Kernel
L2_regularization_S_1 = KernelRidge(kernel='sigmoid', alpha=1.0)
L2_regularization_S_1.fit(x_train, y_train)
S1_y_pred = L2_regularization_S_1.predict(x_test)
S1_mae = mean_absolute_error(y_test, S1_y_pred)
S1_r2 = r2_score(y_test, S1_y_pred)
print(f'  Sigmoid_Kernel & alpha:1  Mean Absolute Error: {S1_mae:.2f}')
print(f'                            R2 Score: {S1_r2:.2f}')
print()
L2_regularization_S_2 = KernelRidge(kernel='sigmoid', alpha=2.0)
L2_regularization_S_2.fit(x_train, y_train)
S2_y_pred = L2_regularization_S_2.predict(x_test)
S2_mae = mean_absolute_error(y_test, S2_y_pred)
S2_r2 = r2_score(y_test, S2_y_pred)
print(f'  Sigmoid_Kernel & alpha:2  Mean Absolute Error: {S2_mae:.2f}')
print(f'                            R2 Score: {S2_r2:.2f}')
```

سپس مقادیر پیش بینی شده توسط مدل را ذخیره و خطای میانگین مطلق و امتیاز R2 را محاسبه میکنیم وسپس چاپ میکنیم.

همین روند را برای مقدار آلفا ۲ تکرار میکنیم.

نتیجه:

```
Linear kernel & alpha:1  Mean Absolute Error: 0.10
                         R2 Score: 0.62

Linear kernel & alpha:2  Mean Absolute Error: 0.10
                         R2 Score: 0.61

PBF kernel & alpha:1  Mean Absolute Error: 0.10
                      R2 Score: 0.68

PBF kernel & alpha:2  Mean Absolute Error: 0.10
                      R2 Score: 0.63

Quadratic_polynomial_Kernel & alpha:1  Mean Absolute Error: 0.08
                                       R2 Score: 0.79

Quadratic_polynomial_Kernel & alpha:2  Mean Absolute Error: 0.09
                                       R2 Score: 0.75

Sigmoid_Kernel & alpha:1  Mean Absolute Error: 0.14
                          R2 Score: 0.22

Sigmoid_Kernel & alpha:2  Mean Absolute Error: 0.15
                          R2 Score: 0.12
```

بهترین کرنل چند جمله ای درجه ۲ با آلفا ۱ بوده است و بازهم کرنل سیگموید نتیجه بسیار ضعیفی داشته است.

نتیجه کرنل چند جمله ای درجه ۲ با L2_Reguralization دارای امتیاز R2 ضعیف تر نسبت به کرنل چند جمله ای درجه ۲ با K_Fold است اما خطای میانگین آن کمتر است.

ج) با تغییر پارامتر Regularization در مقادیر {0.2, 0.8, 1,5, 10, 20, 50, 300} به ازای توابع کرنل خطی، چندجمله‌ای درجه ۲ و ۳ و ۴ و هم چنین RBF مقادیر بهینه را به ازای هر کرنل و هم چنین بهترین کرنل را با بهترین امتیاز R2 به دست بیاورید (از دستور gridsearchcv استفاده نمایید). مقادیر نزدیک صفر برای امتیاز R2 به چه معنا هستند؟

کرنل خطی:

```
param_grid = {'svr__kernel': ['linear'],
              'svr__C': [0.2, 0.8, 1, 5, 10, 20, 50, 300],}
pipe = make_pipeline(StandardScaler(), SVR())
grid_search = GridSearchCV(pipe, param_grid=param_grid, scoring='r2')
grid_search.fit(x, y)
print("  Best parameters of linear kernel:", grid_search.best_params_)
print("  Best r2 score:", grid_search.best_score_)
```

در خط اول با استفاده از دیکشنری param_grid پارامتر های مدل svr برای کرنل خطی که در اینجا مقادیر مختلف C هستند تعریف میشوند.

در خط بعد بنا به نیاز تابع بعدی یک پایپ لاین از کتابخانه sklearn تعریف میشود تا داده ها و مدل SVR را اسکیل بکند.

سپس با استفاده از تابع gridsearchcv از کتابخانه sklearn مقادیر بهینه پارامترها برای بهترین امتیاز R2 محاسبه میشود و سپس چاپ میشوند.

کرنل چند جمله ای:

```
param_grid = {'svr__kernel': [ 'poly'],
              'svr__degree': [2, 3, 4],
              'svr__C': [0.2, 0.8, 1, 5, 10, 20, 50, 300],}
pipe = make_pipeline(StandardScaler(), SVR())
grid_search = GridSearchCV(pipe, param_grid=param_grid, scoring='r2')
grid_search.fit(x, y)
print("  Best parameters of polynomial kernel:", grid_search.best_params_)
print("  Best r2 score:", grid_search.best_score_)
```

در خط اول با استفاده از دیکشنری param_grid پارامتر های مدل svr برای کرنل چند جمله ای که در اینجا مقادیر مختلف C و درجه چند جمله ای هستند تعریف میشوند.

در خط بعد بنا به نیاز تابع بعدی یک پایپ لاین از کتابخانه sklearn تعریف میشود تا داده ها و مدل SVR را اسکیل بکند.

سپس با استفاده از تابع gridsearchcv از کتابخانه sklearn مقادیر بهینه پارامترها برای بهترین امتیاز R2 محاسبه میشود و سپس چاپ میشوند.

کرنل RBF:

```
param_grid = {'svr__kernel': [ 'rbf'],
              'svr__C': [0.2, 0.8, 1, 5, 10, 20, 50, 300],
              'svr__gamma': ['scale', 'auto']}
pipe = make_pipeline(StandardScaler(), SVR())
grid_search = GridSearchCV(pipe, param_grid=param_grid, scoring='r2')
grid_search.fit(x, y)
print("  Best parameters of RBF kernel:", grid_search.best_params_)
print("  Best r2 score:", grid_search.best_score_)
```

در خط اول با استفاده از دیکشنری param_grid پارامتر های مدل svr برای کرنل RBF که در اینجا مقادیر مختلف C و مقدار گاما هستند تعریف میشوند.(مقادیر مختلف گاما بصورت scale و auto)

در خط بعد بنا به نیاز تابع بعدی یک پایپ لاین از کتابخانه sklearn تعریف میشود تا داده ها و مدل SVR را اسکیل بکند.

سپس با استفاده از تابع gridsearchcv از کتابخانه sklearn مقادیر بهینه پارامترها برای بهترین امتیاز R2 محاسبه میشود و سپس چاپ میشوند.

بهترین کرنل با امتیاز R2:

در خط اول با استفاده از دیکشنری param_grid پارامتر های مدل svr برای کرنل های خطی، چند جمله ای و RBF که در اینجا مقادیر مختلف C، مقدار گاما و درجات هستند تعریف میشوند.(مقادیر مختلف گاما بصورت scale و auto)

در خط بعد بنا به نیاز تابع بعدی یک پایپ لاین از کتابخانه sklearn تعریف میشود تا داده ها و مدل SVR را اسکیل بکند.

```
param_grid = {'svr__kernel': ['linear', 'poly', 'rbf'],
              'svr__degree': [2, 3, 4],
              'svr__C': [0.2, 0.8, 1, 5, 10, 20, 50, 300],
              'svr__gamma': ['scale', 'auto']}
pipe = make_pipeline(StandardScaler(), SVR())
grid_search = GridSearchCV(pipe, param_grid=param_grid, scoring='r2')
grid_search.fit(x, y)
print("  Best parameters:", grid_search.best_params_)
print("  Best r2 score:", grid_search.best_score_)
```

سپس با استفاده از تابع gridsearchcv از کتابخانه sklearn مقادیر بهینه پارامترها برای بهترین امتیاز R2 محاسبه میشود و سپس به همراه بهترین کرنل چاپ میشوند.

نتیجه: هرچه امتیاز R2 نزدیکتر به صفر باشد یعنی مدل توانایی پیش بینی درست داده هارا ندارد.

```
Best parameters of linear kernel: {'svr__C': 1, 'svr__kernel': 'linear'}
Best r2 score: -0.18788561004739363

Best parameters of polynomial kernel: {'svr__C': 0.2, 'svr__degree': 3, 'svr__kernel': 'poly'}
Best r2 score: -1.0844978577657334

Best parameters of RBF kernel: {'svr__C': 5, 'svr__gamma': 'auto', 'svr__kernel': 'rbf'}
Best r2 score: 0.36538189286791123

Best parameters: {'svr__C': 5, 'svr__degree': 2, 'svr__gamma': 'auto', 'svr__kernel': 'rbf'}
Best r2 score: 0.36538189286791123
```

د) در نرم افزار متلب به کمک دستور مناسب، فایل داده‌ها (با فرمت Excel) را وارد کرده و رگرسیون غیر خطی را با تابع زیر بر روی داده‌ها اعمال و امتیاز R2 و ریشه‌ی میانگین مربعات خطا را برای تابع برازش شده (به دست آوردن ضرایب مجهول b) به دست آورید.

$$\log(y) = b_1 - b_2.x_1.\exp(-b_3.x_2)$$

پاسخ:

در مرحله اول مسیر فایل اکسل را در filepath ذخیره میکنیم:

سپس filepath تعریف شده را به عنوان یک فایل اکسل با استفاده از تابع xlsread بازخوانی کرده و در dataset ذخیره میکنیم:

```
>> filepath = 'C:\Users\Nima\Desktop\AI Project 2\Qustion2\Performance-Degradation Data Nelson.xlsx';
>> dataset = xlsread(filepath);
```

تقسیم بندی داده های به داده های ورودی و داده های خروجی:

```
>> y = dataset(:,1);
>> x1 = dataset(:,2);
>> x2 = dataset(:,3);
```

داده ها نیاز به نرمال سازی دارند لذا:

```
>> normalized_x1 = normalize(x1, 'range', [0.1, 1]);
>> normalized_x2 = normalize(x2, 'range', [0.1, 1]);
```

در این قسمت از کد با استفاده از تابع normalize مقادیر داده های x1 و x2 به صورت یک بازه از ۰.۱ تا ۱ نرمال سازی شدند.

سپس مقادیر نرمالایز شده x1 و x2 به صورت یک ماتریس دو ستونه x ذخیره میشوند:

```
>> x = [normalized_x1 normalized_x2]
```

داده های خروجی را هم به ترتیب قبل نرمال سازی میکنیم:

```
>> normalized_y = normalize(y, 'range', [0.1, 1]);
```

تعریف یک تابع با متغیر های ماتریسی b و x و ذخیره آن در modelfun:

```
>> modelFun = @(b,x) b(1) - b(2).*x(:,1).*exp(-b(3).*x(:,2));
```

سپس جهت انجام رگرسیون غیر خطی توسط این تابع برروی داده ها متغیر های وابسته و مستقل تابع که normalized_x1,normalized_x2 و normalized_y هستند به همراه تابع تعریف شده بین آنها به همراه مقادیر اولیه فرضی برای ماتریس b وارد تابع fitnlm میکنیم.

```
Non_linear_reg = fitnlm(x,log(normalized_y),modelFun,[1 1 1]);
```

برای نمایش ضرایب بدست آمده برای ارایه های ماتریس b پس از فیت کردن تابع برروی داده از تابع زیر استفاده میشود:

```
>> estimates = Non_linear_reg.Coefficients.Estimate;
>> estimates

estimates =

   -0.2551
    0.0140
   -5.0781
```

ضرایب بدست b عبارتند از :

b1 = - 0.2551

b2 = 0.0140

b3 = - 5.0781

پس تابع مورد نظر به شکل log(y) = - 0.2551 - (0.0140 * x1 * exp (- 5.0781* x2)) در خواهد آمد.

حال میبایست مقادیر R2 و ریشه ی میانگین مربعات خطا را بدست آوریم.

برای این هدف ابتدا ضرایب بدست آمده را در تابع جاگذاری میکنیم و مقدار خروجی را در y_pred ذخیره میکنیم.

```
>> y_pred_by_model = exp(y_pred);
```

خروجی تابع لگاریتم y است برای تبدیل آن به y :

حالا خروجی های مدل رگرسیون ما در y_pred_by_model ذخیره است.

```
>> y_pred = estimates(1) - estimates(2)*x(:,1).*exp(-estimates(3)*x(:,2));
```

پس از تعریف فرمول R2 آنرا چاپ میکنیم:

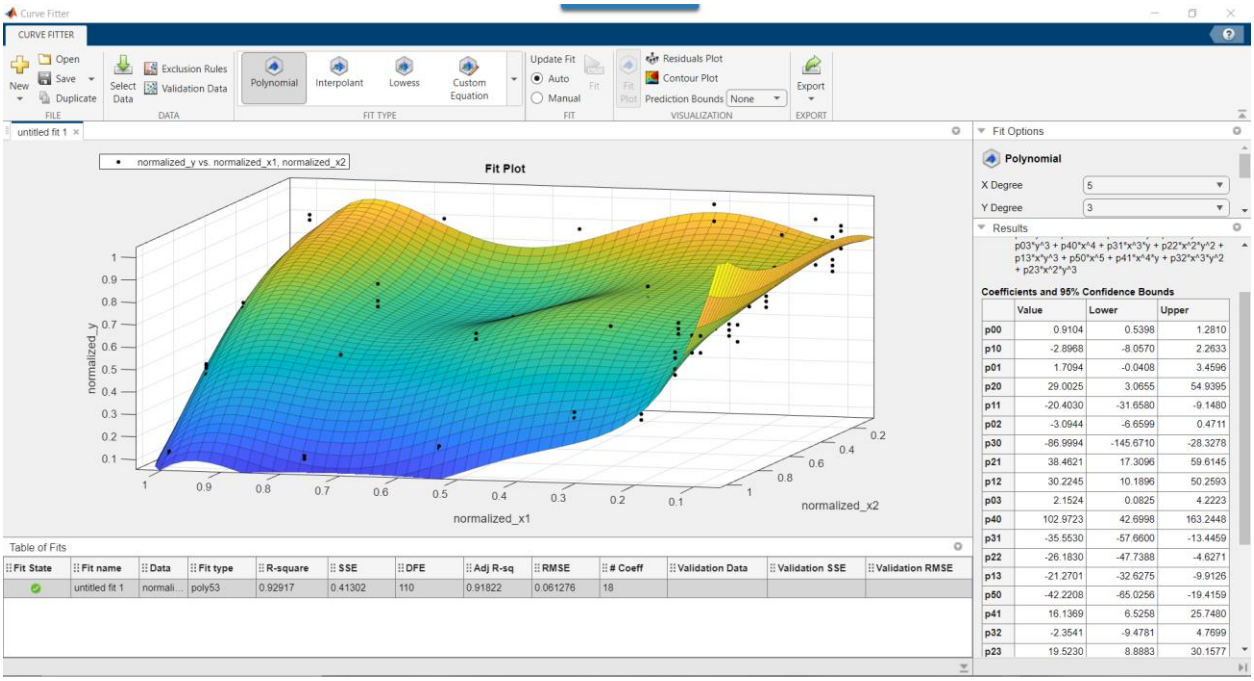
```
>> RSS = sum((normalized_y - y_pred_by_model).^2);
>> TSS = sum((normalized_y - mean(normalized_y)).^2);
>> R2 = 1 - RSS/TSS;
>> R2

R2 =

    0.8703
```

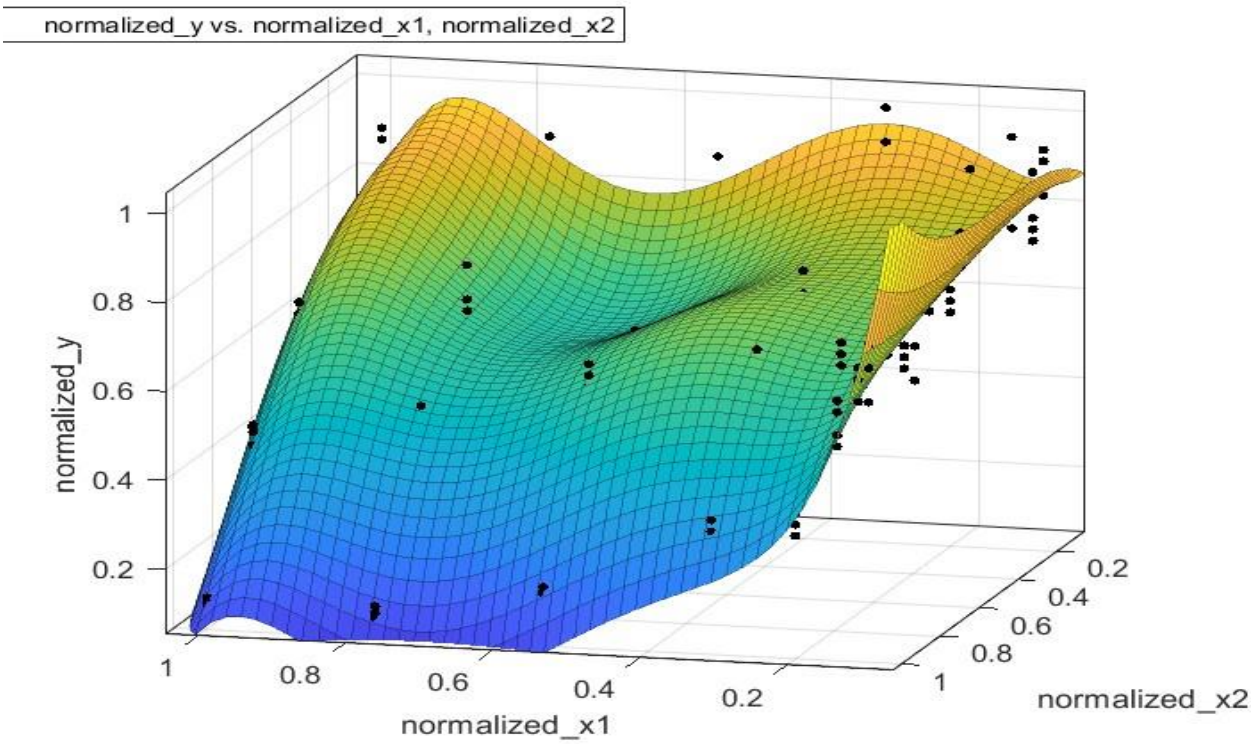
و برای محاسبه ریشه ی میانگین مربعات خطا:

```
>> rmse = rmse(normalized_y, y_pred_by_model);
>> rmse

rmse =

    0.0769
```

ه) به کمک دستور cftool در متلب و با استفاده از داده‌های آزمایش، تابع چندجمله‌ای را به ازای مقادیر مختلف درجات x1 و x2 بر داده‌ها برازش کرده و در حالتی که امتیاز R2 بیشینه می‌شود، مقادیر امتیاز R2 و RMS خطا را به همراه ضرایب چندجمله‌ای و رویه‌ی ایجاد شده ارائه نمایید.

پاسخ:





بهترین تابع از جهت امتیاز R2(R-square) و خطای RMSE : تابعی با x1 با درجه ۵ و x2 با درجه ۳

Polynomial Surface Fit (poly53)

f(x,y) = p00 + p10*x + p01*y + p20*x^2 + p11*x*y + p02*y^2 + p30*x^3 + p21*x^2*y + p12*x*y^2 + p03*y^3 + p40*x^4 + p31*x^3*y + p22*x^2*y^2 + p13*x*y^3 + p50*x^5 + p41*x^4*y + p32*x^3*y^2 + p23*x^2*y^3

Coefficients and 95% Confidence Bounds

	Value	Lower	Upper
p00	0.9104	0.5398	1.2810
p10	-2.8968	-8.0570	2.2633
p01	1.7094	-0.0408	3.4596
p20	29.0025	3.0655	54.9395
p11	-20.4030	-31.6580	-9.1480
p02	-3.0944	-6.6599	0.4711
p30	-86.9994	-145.6710	-28.3278
p21	38.4621	17.3096	59.6145
p12	30.2245	10.1896	50.2593
p03	2.1524	0.0825	4.2223
p40	102.9723	42.6998	163.2448
p31	-35.5530	-57.6600	-13.4459
p22	-26.1830	-47.7388	-4.6271
p13	-21.2701	-32.6275	-9.9126
p50	-42.2208	-65.0256	-19.4159
p41	16.1369	6.5258	25.7480
p32	-2.3541	-9.4781	4.7699
p23	19.5230	8.8883	30.1577

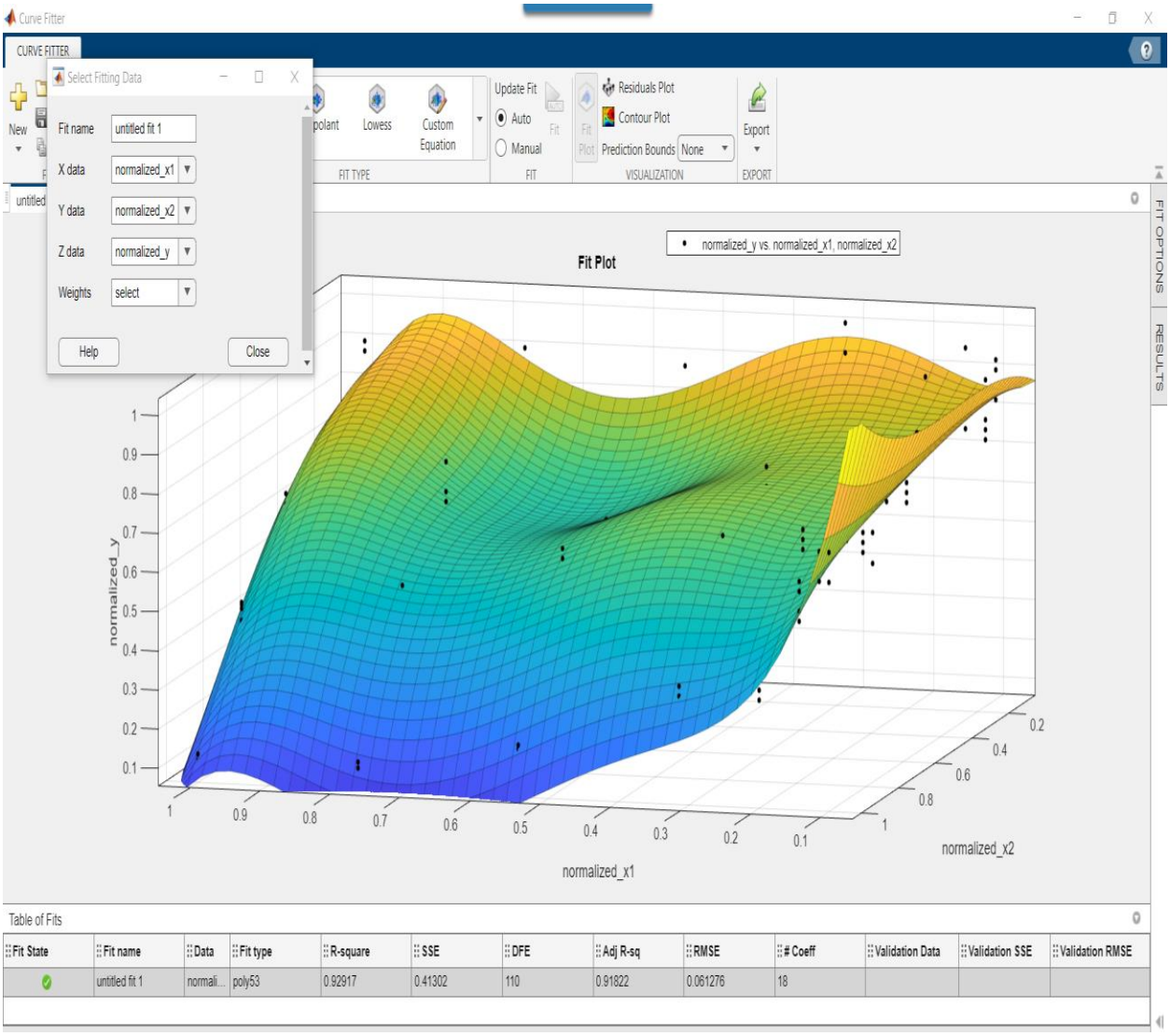


Table of Fits

Fit State	Fit name	Data	Fit type	R-square	SSE	DFE	Adj R-sq	RMSE	# Coeff	Validation Data	Validation SSE	Validation RMSE
✓	untitled fit 1	normali...	poly53	0.92917	0.41302	110	0.91822	0.061276	18			