Sonal Shukla (CS23176)

Lab-7 Assignment-1 : To apply Support Vector Machine for Regression on Students Performance Dataset and analyze the impact of different kernels on model accuracy.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
path="/content/drive/MyDrive/ML_DATASETS/Student_Performance_dataset_5000 - Student_Performance_dataset_5000.csv"
df=pd.read_csv(path)
df.head(5)
```

	student_id	age	gender	academic_level	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_
0	1	18	Other	High School	7.64	1.56	2.20	3.05	
1	2	18	Other	High School	2.21	2.22	2.10	1.65	
2	3	22	Male	High School	3.45	0.00	0.29	1.34	
3	4	17	Other	High School	5.75	2.08	3.01	2.27	
4	5	19	Other	High School	6.83	1.72	3.33	2.65	

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   student_id            5000 non-null   int64
 1   age                   5000 non-null   int64
 2   gender                5000 non-null   object
 3   academic_level        5000 non-null   object
 4   study_hours           5000 non-null   float64
 5   self_study_hours      5000 non-null   float64
 6   online_classes_hours  5000 non-null   float64
 7   social_media_hours    5000 non-null   float64
 8   gaming_hours          5000 non-null   float64
 9   sleep_hours           5000 non-null   float64
 10  screen_time_hours     5000 non-null   float64
 11  exercise_minutes      5000 non-null   int64
 12  caffeine_intake_mg    5000 non-null   int64
 13  part_time_job         5000 non-null   int64
 14  upcoming_deadline     5000 non-null   int64
 15  internet_quality      5000 non-null   object
 16  mental_health_score   5000 non-null   int64
 17  focus_index           5000 non-null   float64
 18  burnout_level         5000 non-null   float64
 19  productivity_score    5000 non-null   float64
 20  exam_score            5000 non-null   float64
dtypes: float64(11), int64(7), object(3)
memory usage: 820.4+ KB
```

```
df.describe()
```

	student_id	age	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_hours	sleep
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.
mean	2500.500000	20.520400	4.539594	2.478734	2.011984	2.998086	1.564514	7.
std	1443.520003	2.870406	1.821665	1.177990	0.983906	1.467949	1.110807	1.
min	1.000000	16.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.
25%	1250.750000	18.000000	3.250000	1.660000	1.320000	1.990000	0.670000	6.
50%	2500.500000	20.000000	4.530000	2.480000	2.010000	2.980000	1.490000	7.
75%	3750.250000	23.000000	5.760000	3.290000	2.690000	4.030000	2.340000	7.
max	5000.000000	25.000000	11.840000	7.410000	6.000000	8.280000	5.640000	10.

```
df.shape
```

```
(5000, 21)
```

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

	0
student_id	0
age	0
gender	0
academic_level	0
study_hours	0
self_study_hours	0
online_classes_hours	0
social_media_hours	0
gaming_hours	0
sleep_hours	0
screen_time_hours	0
exercise_minutes	0
caffeine_intake_mg	0
part_time_job	0
upcoming_deadline	0
internet_quality	0
mental_health_score	0
focus_index	0
burnout_level	0
productivity_score	0
exam_score	0

dtype: int64

```
df.columns
```

```
Index(['student_id', 'age', 'gender', 'academic_level', 'study_hours',
       'self_study_hours', 'online_classes_hours', 'social_media_hours',
       'gaming_hours', 'sleep_hours', 'screen_time_hours', 'exercise_minutes',
       'caffeine_intake_mg', 'part_time_job', 'upcoming_deadline',
       'internet_quality', 'mental_health_score', 'focus_index',
       'burnout_level', 'productivity_score', 'exam_score'],
      dtype='object')
```

```
df.columns = df.columns.str.replace(' ', '_')
df.head()
```

	student_id	age	gender	academic_level	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_
0	1	18	Other	High School	7.64	1.56	2.20	3.05	
1	2	18	Other	High School	2.21	2.22	2.10	1.65	
2	3	22	Male	High School	3.45	0.00	0.29	1.34	
3	4	17	Other	High School	5.75	2.08	3.01	2.27	
4	5	19	Other	High School	6.83	1.72	3.33	2.65	

5 rows × 21 columns

```
df.rename(columns={'Mean_of_the_integrated_profile': 'Mean_IP', 'Standard_deviation_of_the_integrated_profile': 'Std_Dev_IP
```

```
df.columns
```

```
Index(['student_id', 'age', 'gender', 'academic_level', 'study_hours',
       'self_study_hours', 'online_classes_hours', 'social_media_hours',
       'gaming_hours', 'sleep_hours', 'screen_time_hours', 'exercise_minutes',
       'caffeine_intake_mg', 'part_time_job', 'upcoming_deadline',
       'internet_quality', 'mental_health_score', 'focus_index',
       'burnout_level', 'productivity_score', 'exam_score'],
      dtype='object')
```

```
X = df.drop(columns=['exam_score'])
y = df['exam_score']
```

```
X.head()
```

	student_id	age	gender	academic_level	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_
0	1	18	Other	High School	7.64	1.56	2.20	3.05	
1	2	18	Other	High School	2.21	2.22	2.10	1.65	
2	3	22	Male	High School	3.45	0.00	0.29	1.34	
3	4	17	Other	High School	5.75	2.08	3.01	2.27	
4	5	19	Other	High School	6.83	1.72	3.33	2.65	

```
X = pd.get_dummies(X,dtype = int)
```

```
X.head()
```

	student_id	age	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_hours	sleep_hours	scre
0	1	18	7.64	1.56	2.20	3.05	2.19	6.52	
1	2	18	2.21	2.22	2.10	1.65	2.55	5.97	
2	3	22	3.45	0.00	0.29	1.34	2.08	8.39	
3	4	17	5.75	2.08	3.01	2.27	2.20	6.31	
4	5	19	6.83	1.72	3.33	2.65	0.70	8.01	

5 rows × 26 columns

```
X = df.iloc[:,3:-1]
X = pd.get_dummies(X, dtype=int)
X
```

	study_hours	self_study_hours	online_classes_hours	social_media_hours	gaming_hours	sleep_hours	screen_time_hours
0	7.64	1.56	2.20	3.05	2.19	6.52	6.47
1	2.21	2.22	2.10	1.65	2.55	5.97	6.05
2	3.45	0.00	0.29	1.34	2.08	8.39	7.62
3	5.75	2.08	3.01	2.27	2.20	6.31	11.67
4	6.83	1.72	3.33	2.65	0.70	8.01	10.02
...	...	...	...	...	...	...	...
4995	3.93	1.42	0.54	2.58	2.62	9.30	6.38
4996	4.00	0.00	3.00	4.45	2.47	6.42	7.32
4997	5.63	2.52	2.89	3.58	1.07	6.65	9.51
4998	3.58	2.83	1.53	5.88	1.28	7.36	8.31
4999	5.60	4.14	2.17	3.05	2.74	6.38	9.59

5000 rows × 21 columns

```
y=df.iloc[:,-1]
print(y)
```

```
0       50.16
1        1.00
2       18.30
3        9.37
4       27.81
        ...
4995    22.51
4996     2.00
4997    27.89
4998     3.63
4999    10.76
Name: exam_score, Length: 5000, dtype: float64
```

```
X.shape
```

```
(5000, 21)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
X_train.shape,X_test.shape
```

```
((3500, 21), (1500, 21))
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[-0.32898634,  0.33191255, -1.72603855, ..., -0.70045009,
         1.37310163, -0.69275929],
       [ 1.3038051 , -1.21217512,  1.60778132, ..., -0.70045009,
         1.37310163, -0.69275929],
       [ 0.838076  ,  0.90567191,  2.75283191, ..., -0.70045009,
        -0.72827821,  1.44350284],
       ...,
       [ 1.09011763,  0.35722547,  1.11125495, ..., -0.70045009,
        -0.72827821,  1.44350284],
       [-0.21940302, -2.08968943, -0.75325221, ..., -0.70045009,
        -0.72827821,  1.44350284],
       [-0.32898634, -1.26280095, -0.24659265, ..., -0.70045009,
        -0.72827821,  1.44350284]])
```

```
y_train
```

	exam_score
1840	10.53
2115	35.73
4437	25.82
1146	20.95
2486	27.36
...	...
4426	10.57
466	23.77
3092	25.68
3772	17.15
860	2.81

3500 rows × 1 columns

dtype: float64

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# Initialize SVR with default hyperparameters
svr = SVR()

# Fit the model
svr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svr.predict(X_test)

# Calculate R-squared and Mean Squared Error
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f'Model R-squared score with default hyperparameters: {r2:.4f}')
print(f'Model Mean Squared Error with default hyperparameters: {mse:.4f}')
```

```
Model R-squared score with default hyperparameters: 0.7964
Model Mean Squared Error with default hyperparameters: 28.8250
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with linear kernel and C=1.0
linear_svr_1=SVR(kernel='linear',C=1.0)
linear_svr_1.fit(X_train,y_train)
y_pred_linear_1=linear_svr_1.predict(X_test)

r2_linear_1 = r2_score(y_test, y_pred_linear_1)
mse_linear_1 = mean_squared_error(y_test, y_pred_linear_1)

print(f'Model R-squared score with linear kernel and C=1.0 : {r2_linear_1:.4f}')
print(f'Model Mean Squared Error with linear kernel and C=1.0 : {mse_linear_1:.4f}')
```

```
Model R-squared score with linear kernel and C=1.0 : 0.8192
Model Mean Squared Error with linear kernel and C=1.0 : 25.6011
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with linear kernel and C=100.0
linear_svr_100=SVR(kernel='linear',C=100.0)
linear_svr_100.fit(X_train,y_train)
y_pred_linear_100=linear_svr_100.predict(X_test)

r2_linear_100 = r2_score(y_test, y_pred_linear_100)
mse_linear_100 = mean_squared_error(y_test, y_pred_linear_100)

print(f'Model R-squared score with linear kernel and C=100.0 : {r2_linear_100:.4f}')
print(f'Model Mean Squared Error with linear kernel and C=100.0 : {mse_linear_100:.4f}')
```

```
Model R-squared score with linear kernel and C=100.0 : 0.8192
Model Mean Squared Error with linear kernel and C=100.0 : 25.5979
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with linear kernel and C=1000.0
linear_svr_1000=SVR(kernel='linear',C=1000.0)
linear_svr_1000.fit(X_train,y_train)
y_pred_linear_1000=linear_svr_1000.predict(X_test)

r2_linear_1000 = r2_score(y_test, y_pred_linear_1000)
mse_linear_1000 = mean_squared_error(y_test, y_pred_linear_1000)

print(f'Model R-squared score with linear kernel and C=1000.0 : {r2_linear_1000:.4f}')
print(f'Model Mean Squared Error with linear kernel and C=1000.0 : {mse_linear_1000:.4f}')
```

```
Model R-squared score with linear kernel and C=1000.0 : 0.8192
Model Mean Squared Error with linear kernel and C=1000.0 : 25.5975
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with polynomial kernel and C=1.0
poly_svr_1=SVR(kernel='poly',C=1.0)
poly_svr_1.fit(X_train,y_train)
y_pred_poly_1=poly_svr_1.predict(X_test)

r2_poly_1 = r2_score(y_test, y_pred_poly_1)
mse_poly_1 = mean_squared_error(y_test, y_pred_poly_1)

print(f'Model R-squared score with polynomial kernel and C=1.0 : {r2_poly_1:.4f}')
print(f'Model Mean Squared Error with polynomial kernel and C=1.0 : {mse_poly_1:.4f}')
```

```
Model R-squared score with polynomial kernel and C=1.0 : 0.7723
Model Mean Squared Error with polynomial kernel and C=1.0 : 32.2382
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with polynomial kernel and C=100.0
poly_svr_100=SVR(kernel='poly',C=100.0)
poly_svr_100.fit(X_train,y_train)
y_pred_poly_100=poly_svr_100.predict(X_test)

r2_poly_100 = r2_score(y_test, y_pred_poly_100)
mse_poly_100 = mean_squared_error(y_test, y_pred_poly_100)

print(f'Model R-squared score with polynomial kernel and C=100.0 : {r2_poly_100:.4f}')
print(f'Model Mean Squared Error with polynomial kernel and C=100.0 : {mse_poly_100:.4f}')
```

```
Model R-squared score with polynomial kernel and C=100.0 : 0.7046
Model Mean Squared Error with polynomial kernel and C=100.0 : 41.8269
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with polynomial kernel and C=1000.0
poly_svr_1000=SVR(kernel='poly',C=1000.0)
poly_svr_1000.fit(X_train,y_train)
y_pred_poly_1000=poly_svr_1000.predict(X_test)

r2_poly_1000 = r2_score(y_test, y_pred_poly_1000)
mse_poly_1000 = mean_squared_error(y_test, y_pred_poly_1000)

print(f'Model R-squared score with polynomial kernel and C=1000.0 : {r2_poly_1000:.4f}')
print(f'Model Mean Squared Error with polynomial kernel and C=1000.0 : {mse_poly_1000:.4f}')
```

```
Model R-squared score with polynomial kernel and C=1000.0 : 0.6608
Model Mean Squared Error with polynomial kernel and C=1000.0 : 48.0268
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error
```

```
Model R-squared score with sigmoid kernel and C=1.0 : 0.7725
Model Mean Squared Error with sigmoid kernel and C=1.0 : 32.2070
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with sigmoid kernel and C=100.0
sigmoid_svr_100=SVR(kernel='sigmoid',C=100.0)
sigmoid_svr_100.fit(X_train,y_train)
y_pred_sigmoid_100=sigmoid_svr_100.predict(X_test)

r2_sigmoid_100 = r2_score(y_test, y_pred_sigmoid_100)
mse_sigmoid_100 = mean_squared_error(y_test, y_pred_sigmoid_100)

print(f'Model R-squared score with sigmoid kernel and C=100.0 : {r2_sigmoid_100:.4f}')
print(f'Model Mean Squared Error with sigmoid kernel and C=100.0 : {mse_sigmoid_100:.4f}')
```

```
Model R-squared score with sigmoid kernel and C=100.0 : -1132.2602
Model Mean Squared Error with sigmoid kernel and C=100.0 : 160448.8402
```

```
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# SVR with sigmoid kernel and C=1000.0
sigmoid_svr_1000=SVR(kernel='sigmoid',C=1000.0)
sigmoid_svr_1000.fit(X_train,y_train)
y_pred_sigmoid_1000=sigmoid_svr_1000.predict(X_test)

r2_sigmoid_1000 = r2_score(y_test, y_pred_sigmoid_1000)
mse_sigmoid_1000 = mean_squared_error(y_test, y_pred_sigmoid_1000)
```