Deakin University

SIG720- OnTrack Submission

Task C1

Submitted by
HARPREET SINGH
223925166
Attempt #2
06-08-2023

Target Grade : CREDIT

Task Details -
Part-1
Data Description: This dataset contains 206 attributes of 70 children with physical and motor disability based on ICF-CY.

1. Determine the number of subgroups from the dataset using attributes 3 to 205 i.e., exclude attributes 1, 2 and 206. Is this number the same as the number of classes presented by attribute 206? Explain and justify your findings.
2. Is this data facing the curse of dimensionality? If so, then how to solve this problem. Explain with a two-dimensional plot and report relevant loss of information.
3. After applying principal component analysis (PCA) on a given dataset, it was found that the percentage of variance for the first N components is X%. How is this percentage of variance computed?
Dataset filename: obesity_levels.csv

Part-2
Dataset description: This dataset includes data for the estimation of obesity levels in individuals based on their eating habits and physical condition. The data contains 17 attributes and 2111 records.

4. Create a machine learning (ML) model for predicting “weight” using all features except “NObeyesdad” and report observed performance. Explain your results based on following criteria:
a. What model have you selected for solving this problem and why?
b. Have you made any assumption for the target variable? If so, then why?
c. What have you done with text variables? Explain.
d. Have you optimized any model parameters? What is the benefit of this action?
e. Have you applied any steps for handling overfitting or underfitting issues? What is that?

Part-1

Data Description: This dataset contains 206 attributes of 70 children with physical and motor disability based on ICF-CY.

1. Determine the number of subgroups from the dataset using attributes 3 to 205 i.e., exclude attributes 1, 2 and 206. Is this number the same as the number of classes presented by attribute 206? Explain and justify your findings.

The given dataset contains 206 attributes of 70 children with physical and motor disability based on ICF-CY.

```
[10]: sc = pd.read_csv('SCADI.csv')
      print(f"Shape of the Data:{sc.shape}\nNumber of Records:{sc.shape[0]}\nNumber of features:{sc.shape[1]}")
      sc.head()
```

```
Shape of the Data:(70, 206)
Number of Records:70
Number of features:206
```

[10]:

Gender	Age	d 5100-0	d 5100-1	d 5100-2	d 5100-3	d 5100-4	d 5100-8	d 5100-9	d 5101-0	...	d 57022-8	d 57022-9	d 571-0	d 571-1	d 571-2	d 571-3	d 571-4	d 571-8	d 571-9	Classes
0	18	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	1	0	0	class6
0	22	0	0	0	0	1	0	0	0	...	0	0	0	0	0	1	0	0	0	class6
0	18	0	0	0	1	0	0	0	0	...	0	0	0	0	0	1	0	0	0	class6
1	18	0	0	0	0	1	0	0	0	...	0	0	0	0	1	0	0	0	0	class6
0	19	0	0	0	0	1	0	0	0	...	0	0	0	0	1	0	0	0	0	class6

ows × 206 columns

In Classes column of the above dataset, there are 7 different classes/labels which are described in the dataset

```
[9]: print(f"Unique labels in classes column: {sc['Classes'].unique()}")
     print(f"Unique number of labels in classes column: {sc['Classes'].nunique()}")
```

```
Unique labels in classes column: ['class6' 'class2' 'class4' 'class7' 'class1' 'class5' 'class3']
Unique number of labels in classes column: 7
```

In further analysis, we are checking if based on the number of features and Records, we can get the same number of classes/ groups or not, using statistical techniques such as:
K Means Clustering Algorithm

K-means clustering is a method for grouping n observations into K clusters. It uses vector quantization and aims to assign each observation to the cluster with the nearest mean or centroid, which serves as a prototype for the cluster. K-means clustering is now widely used in machine learning to partition data points into K clusters based on their similarity. The goal is to minimize the sum of squared distances between the data points and their corresponding cluster centroids, resulting in clusters that are internally homogeneous and distinct from each other

To Further drill down, First the relevant features are taken from the dataset excluding Gender,Age & Classes

```
[8]: X = sc.drop(columns = ['Gender','Age','Classes'])
     X.shape
```

```
[8]: (70, 203)
```

After that the data is scaled using Standard Scaler

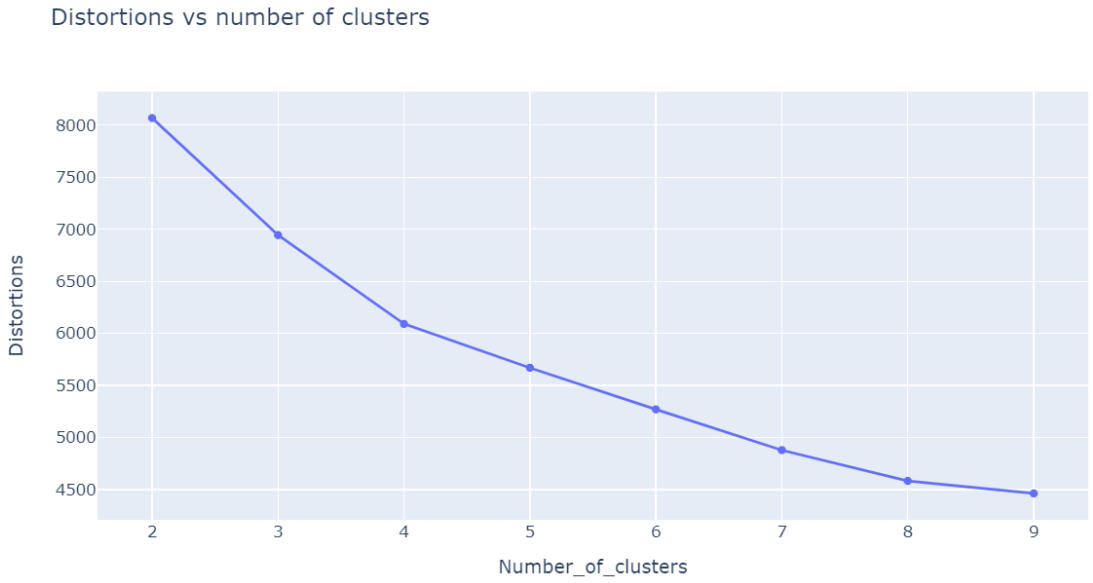
```
[17]: scaler = StandardScaler()
      scaled_X = scaler.fit_transform(X)
      scaled_X
```

```
[17]: array([[-0.2773501 , -0.40824829, -0.61036794, ...,  3.26598632,
                0.        ,  0.        ],
              [-0.2773501 , -0.40824829, -0.61036794, ..., -0.30618622,
                0.        ,  0.        ],
              [-0.2773501 , -0.40824829, -0.61036794, ..., -0.30618622,
                0.        ,  0.        ],
              ...,
              [-0.2773501 , -0.40824829, -0.61036794, ..., -0.30618622,
                0.        ,  0.        ],
              [-0.2773501 , -0.40824829, -0.61036794, ..., -0.30618622,
                0.        ,  0.        ],
              [-0.2773501 , -0.40824829, -0.61036794, ..., -0.30618622,
                0.        ,  0.        ]])
```

Now the data is ready to be applied by K Means algorithm.
For this we run the algorithm for different number of clusters and to find the optimal number of clusters ,we use elbow method,which helps to determine the optimal number of clusters

```
[18]: wcss = [] # empty list to store distortions for differnt number of clusters
      values = range(2,10)
      li = []
      for k in values:
          kmm = KMeans(n_clusters=k,random_state=42)
          kmm.fit(scaled_X)
          wcss.append(kmm.inertia_)
          li.append(f"For n_clusters = {k},inertia :{kmm.inertia_}")
```

```
[19]: fig = px.line(x=values, y=wcss,title = 'Distortions vs number of clusters',
                   labels={"x":"Number_of_clusters","y":"Distortions"},
                   height=500,width=900,markers=True)
      fig.show()
```




```
[12]: li
```

```
[12]: ['For n_clusters = 2,inertia :8067.6498830395285',
       'For n_clusters = 3,inertia :6944.268517928714',
       'For n_clusters = 4,inertia :6091.816110772894',
       'For n_clusters = 5,inertia :5670.663840404521',
       'For n_clusters = 6,inertia :5271.766858895061',
       'For n_clusters = 7,inertia :4880.495550659746',
       'For n_clusters = 8,inertia :4584.945327728501',
       'For n_clusters = 9,inertia :4464.50887460046']
```

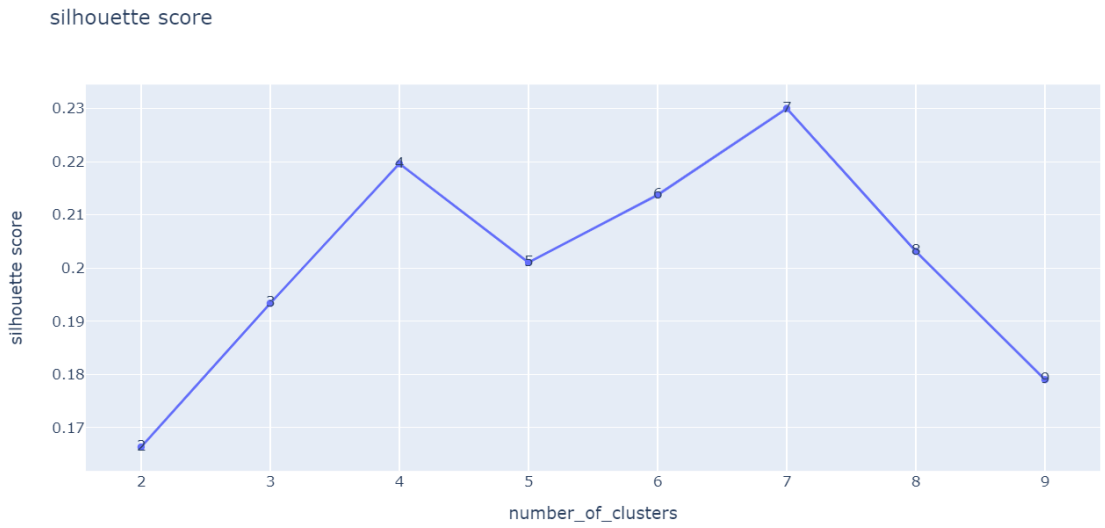
From the above Graph and list, It looks as the elbow is made at Cluster : 4 with mean distortion = 6091.81 and Cluster : 7 has mean distortion = 4880.49,
With this in mind we could go for clusters 4 or cluster 7 as there is significant decrease of distortion from cluster 3 to cluster 4 and there is not much decrease in distortions between cluster 5 and cluster 4.
Similarly From Cluster 6 to cluster 7 there is significant decrease in distortion but not much decrease in distortion from cluster 7 to cluster 8

So We can have clusters 4 or clusters 7.To check further we will go for Silhouette Score and the clusters which will have maximum score can be considered the final cluster

[20]:
```
sl = []
# values = range(2,50)
for k in values:
    kmeans = KMeans(n_clusters=k,random_state=42)
    predict = kmeans.fit_predict(scaled_X)
    score = silhouette_score(scaled_X,predict,random_state=1)
    sl.append(score)
    print(f"For n_clusters = {k},silhouette score :{score}")
```

```
For n_clusters = 2,silhouette score :0.1663380133050078
For n_clusters = 3,silhouette score :0.19336960896484867
For n_clusters = 4,silhouette score :0.21962896326983072
For n_clusters = 5,silhouette score :0.2010513972240522
For n_clusters = 6,silhouette score :0.21375517781568876
For n_clusters = 7,silhouette score :0.2300238873038162
For n_clusters = 8,silhouette score :0.20313742323883902
For n_clusters = 9,silhouette score :0.17900613645430866
```

[21]:
```
fig = px.line(x=values, y=sl,title = 'silhouette score',
              labels={"x":"number_of_clusters","y":"silhouette score"},
              height=500,width=1000,text=values)
fig.show()
```



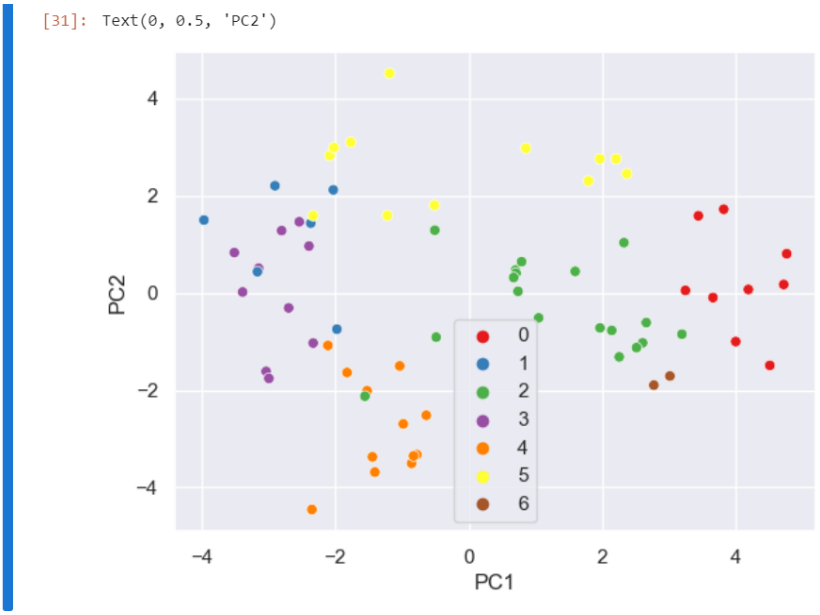
From the above Graph, We can see that there are 2 peaks in the graph for cluster 4 and cluster 7 respectively
CLuster 7 has a score of 0.23
Cluster 4 has a score of 0.21
This shows that we are more inclined to choose 7 clusters or 7 classes for our problem
And the problem statement with the given dataset has stated 7 classes itself.

Final Interpretation:--
Initially we have been provided with a dataset that contains 206 attributes of 70 children with physical and motor disability based on ICF-CY.
This data is segmented into 7 different classes based on 206 features and only 70 records
In the modelling technique used, our aim is to divide the children with given

features to subgroup them into various classes/segments and able to check the result that is already given in the dataset i.e. the number of 7 classes or not using clustering techniques

Here , Kmeans clustering is used in modelling and to find the optimal number of classes /groups/segments that will be defined by K Means Algorithm,
Further to determine optimal number of clusters,elbow & Silhouette score method is used to find the final clusters
From above , the K Means algorithm is able to create same number of 7 classes/clusters as it is given in the problem

Below is the clustering used using K Means on Reduced set of features using Principal components



2. Is this data facing the curse of dimensionality? If so, then how to solve this problem. Explain with a two-dimensional plot and report relevant loss of information.

Yes , the data is facing the curse of dimensionality as the number of records are greater than the number of features.Due to this Some of the difficulties that come with high dimensional data manifest during analyzing or visualizing the data to identify patterns, and some manifest while training machine learning models. The difficulties related to training machine learning models due to high dimensional data are basically known as the 'Curse of Dimensionality'

```
[15]: print(f"Shape of the dataset:{sc.shape}")
      print(f"Number of Records in the dataset:{sc.shape[0]}")
      print(f"Number of Features in the dataset:{sc.shape[1]}")
```

```
Shape of the dataset:(70, 206)
Number of Records in the dataset:70
Number of Features in the dataset:206
```

There are basically two facets of Curse of dimensionality

Data Sparsity-,
the available training samples may not have observed targets for all combinations of the attributes. This is because some combinations occur more often than others. Due to this, the training samples available for building the model may not capture all possible combinations. This aspect, where the training samples do not capture all combinations, is referred to as 'Data sparsity' or simply 'sparsity' in high dimensional data
Training a model with sparse data could lead to high-variance or overfitting conditions. This is because while training the model, the model has learnt from the frequently occurring combinations of the attributes and can predict the outcome accurately. In real-time when less frequently occurring combinations are fed to the model, it may not predict the outcome accurately.

Distance Concentration
Distance concentration refers to the problem of all the pairwise distances between different samples/points in the space converging to the same value as the dimensionality of the data increases. Several machine learning models such as clustering or nearest neighbours' methods use distance-based metrics to identify similarities or proximity of the samples. Due to distance concentration, the concept of proximity or similarity of the samples may not be qualitatively relevant in higher dimensions.

To mitigate the problems associated with high dimensional data a suite of techniques generally referred to as 'Dimensionality reduction techniques are used. Dimensionality reduction techniques fall into one of the two categories-
'Feature selection' or 'Feature extraction

In feature selection techniques, the attributes are tested for their worthiness and then selected or eliminated.

Low Variance filter:
In this technique, the variance in the distribution of all the attributes in a dataset is compared and attributes with very low variance are eliminated. Attributes that do not have such much variance will assume an almost constant value and do not contribute to the predictability of the model.

High Correlation filter:

In this technique, the pair wise correlation between attributes is determined. One of the attributes in the pairs that show very high correlation is eliminated and the other retained. The variability in the eliminated attribute is captured through the retained attribute.

Multicollinearity:
In some cases, the high correlation may not be found for pairs of attributes but if each attribute is regressed as a function of others, we may see that variability of some of the attributes are completely captured by the others. This aspect is referred to as multicollinearity and Variance Inflation Factor (VIF) is a popular technique used to detect multicollinearity. Attributes with high VIF values, generally greater than 10, are eliminated.

Feature Ranking:
Decision Tree models such as CART can rank the attributes based on their importance or contribution to the predictability of the model. In high dimensional data, some of the lower ranked variables could be eliminated to reduce the dimensions

In feature extraction techniques, the high dimensional attributes are combined in low dimensional components (PCA or ICA) or factored into low dimensional factors (FA).

Principal Component Analysis, or PCA,
is a dimensionality-reduction technique in which high dimensional correlated data is transformed to a lower dimensional set of uncorrelated components, referred to as principal components. The lower dimensional principle components capture most of the information in the high dimensional dataset. An ‘n’ dimensional data is transformed into ‘n’ principle components and a subset of these ‘n’ principle components is selected based on the percentage of variance in the data intended to be captured through the principle components

In order to solve the problem, we apply various above techniques to reduce the dimensionality of the dataset

```
159]: x.head()
```

159]:

	d 5100-0	d 5100-1	d 5100-2	d 5100-3	d 5100-4	d 5100-8	d 5100-9	d 5101-0	d 5101-1	d 5101-2	...	d 57022-4	d 57022-8	d 57022-9	d 571-0	d 571-1	d 571-2	d 571-3	d 571-4	d 571-8	5
0	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	
1	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	
2	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	
3	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	

5 rows × 203 columns

Above data will be checked for data sparsity as the data matrix contains lots of 0 values,constant value features(low variance features) and further for correlated features to reduce the dimensionality
For this we use feature_engine library and create a pipeline to drop constant features, correlated features , duplicated features etc. to reduce the dimensionality of the dataset

[122]:
```
from feature_engine.selection import DropConstantFeatures,DropDuplicateFeatures,SmartCorrelatedSelection,Drop
from sklearn.pipeline import Pipeline
```

[146]:
```
pipe1 = Pipeline([('constant',DropConstantFeatures(tol=0.99)),
                  ('duplicate',DropDuplicateFeatures()),
                  ('correlation',DropCorrelatedFeatures(threshold=0.5)),
                  ('scaler',StandardScaler()),
                  ])

pipe1.fit(X)
```

[146]:
```
Pipeline(steps=[('constant', DropConstantFeatures(tol=0.99)),
                ('duplicate', DropDuplicateFeatures()),
                ('correlation', DropCorrelatedFeatures(threshold=0.5)),
                ('scaler', StandardScaler())])
```

[147]:
```
print(f"Constant or Quassi constant features to drop:-{len(pipe1.named_steps.constant.features_to_drop_)}")
# pipe1.named_steps.constant.features_to_drop_
```

```
Constant or Quassi constant features to drop:-63
```

[148]:
```
print(f"Duplicate features to drop:-{len(pipe1.named_steps.duplicate.features_to_drop_)}")
```

```
Duplicate features to drop:-27
```

[161]:
```
print(f"Correlated features to drop:-{len(pipe1.named_steps.correlation.features_to_drop_)}")
pipe1.named_steps.correlation.features_to_drop_
```

```
Correlated features to drop:-70
```

[161]:
```
{'d 5101-1',
 'd 5101-4',
 'd 571-4'}
```

[163]:
```
print(f"Before applying techniques used above to reduce dimensionality,\nshape :{X.shape}")
```

```
Before applying techniques used above to reduce dimensionality,
shape :(70, 203)
```

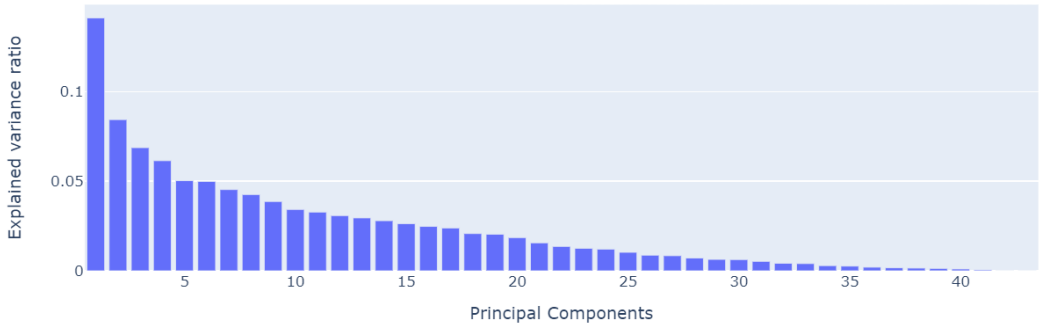
[164]:
```
# applying the above dimensionalty reduction technique
X_reduce = pd.DataFrame(pipe1.transform(X))
print(f"After applying techniques used above to reduce dimensionality,\nshape :{X_reduce.shape}")
```

```
After applying techniques used above to reduce dimensionality,
shape :(70, 43)
```

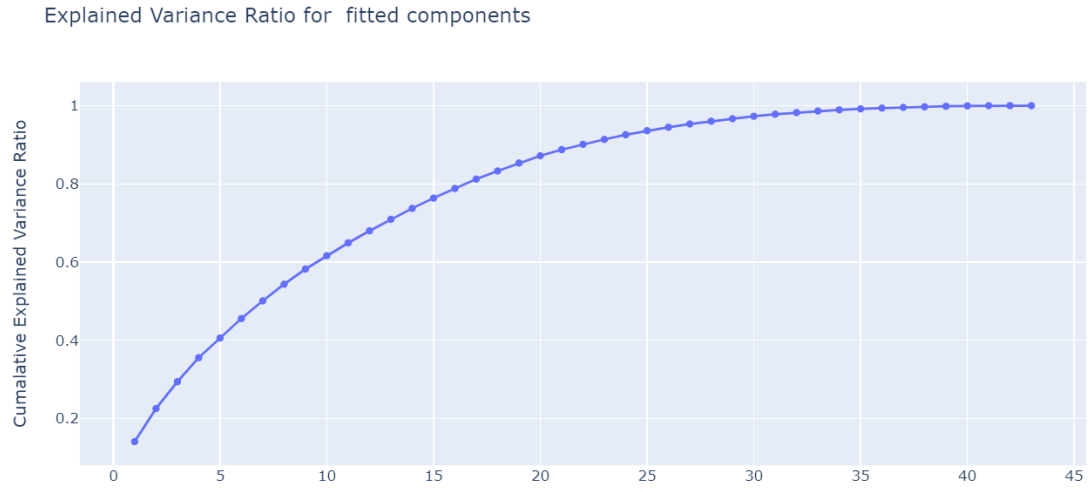
So, By looking constant variance, correlated features, we are able to reduce the dimensionality to 43 prior to 203

Now the dataset will be evaluated using feature extraction technique called PCA to further reduce dimensionality

[166]:
```
# Applying pca on reduce feature dataset
pca = PCA()
pca.fit(X_reduce)
fig = px.bar(x=[i + 1 for i in range(len(pca.explained_variance_ratio_))],
             y=pca.explained_variance_ratio_,labels={"x":"Principal Components","y":"Explained variance ratio"})
fig.show()
```



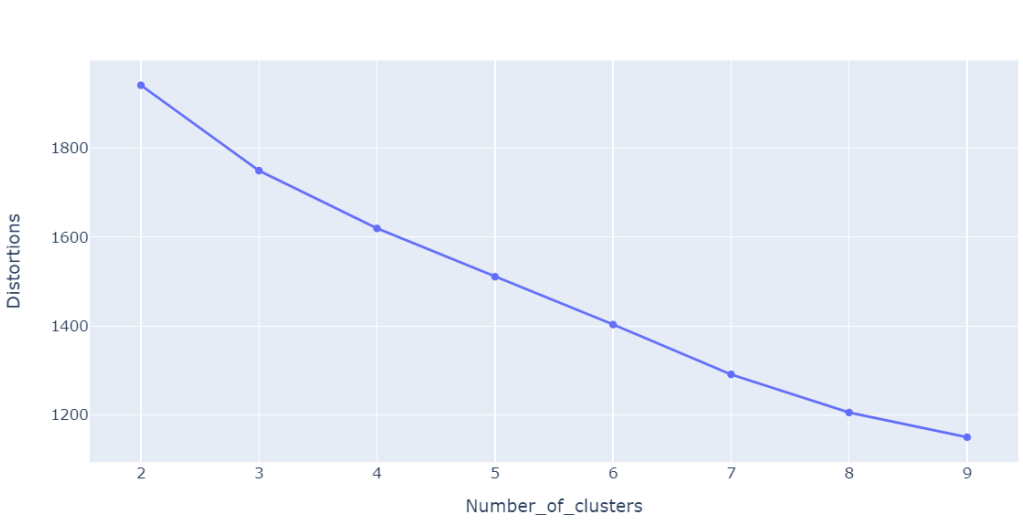
[167]:
```
fig = px.line(x=[i + 1 for i in range(len(pca.explained_variance_ratio_))],
              y=pca.explained_variance_ratio_.cumsum(),title = 'Explained Variance Ratio for \nfitted components',
              labels={"x":"Number of Components","y":"Cumulative Explained Variance Ratio"},height=500,width=1000,markers=True
              )
fig.show()
```

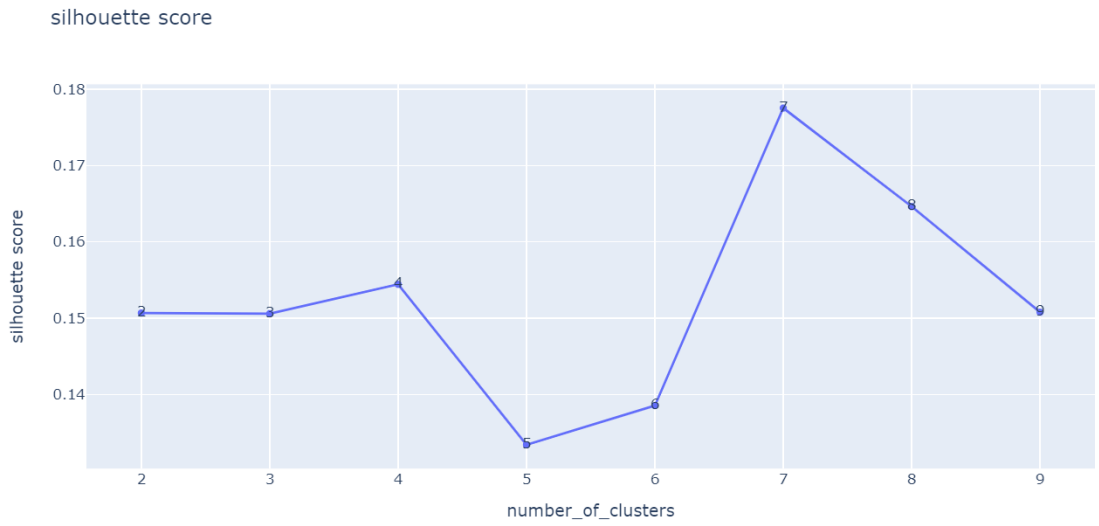


It is recommended that approx 80% variance of data should be preserved and from above line plot , 15 PCA components are able to capture that much variance

So we reduce dimensionality further by projecting the above reduced 43 features into 15 Principal components

[175]:
```
pca_15 = PCA(n_components=15,random_state = 42)
pca_15_transform_df = pca_15.fit_transform(X_reduce)

wcss = [] # empty list to store distortions fr fdiffernt number of clusters
values = range(2,10)
li = []
for k in values:
    kmm = KMeans(n_clusters=k,random_state=42)
    kmm.fit(pca_15_transform_df)
    wcss.append(kmm.inertia_)
    li.append(f"For n_clusters = {k},inertia :{kmm.inertia_}")



fig = px.line(x=values, y=wcss,title = 'Distortions vs number of clusters',
              labels={"x":"Number_of_clusters","y":"Distortions"},
              height=500,width=900,markers=True)
fig.show()
```





From analysis by looking the graphs above we can see that after reducing our features from 203 to just 15
we are getting the same result of 7 clusters (considering the elbow method and Silhouette score which points to 7 here)

Here we lost 20%(using 80% variance in data) of the information and still got the decent result

Now KMeans is applied on the reduced feature dataset to find the optimal number of segments/clusters

[189]:
```
# Applying KMEANS clustering to reduced feature dataset
kmean = KMeans(n_clusters=7)
labels = kmean.fit_predict(pca_15_transform_df)
labels
```

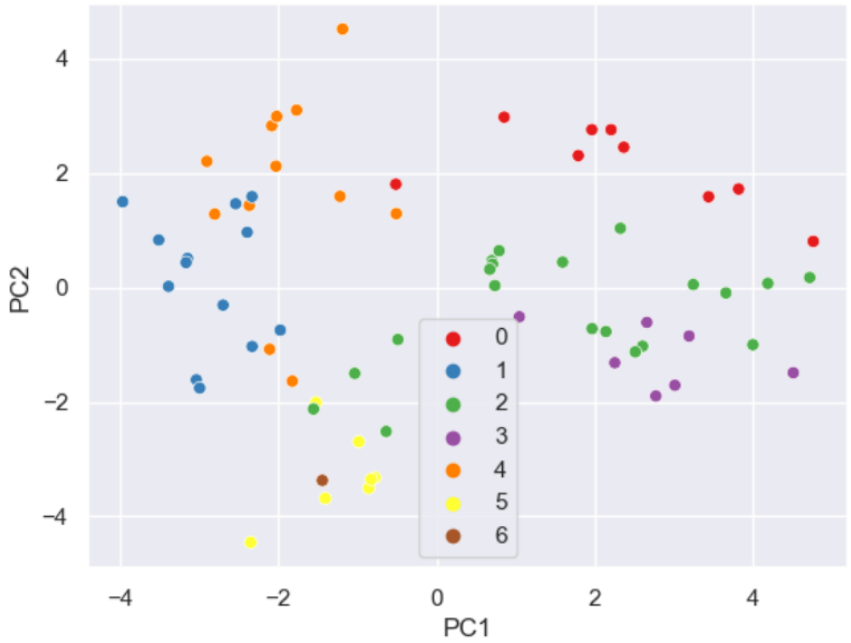
[189]:
```
array([3, 2, 0, 2, 4, 6, 3, 4, 4, 5, 0, 4, 0, 5, 0, 3, 1, 2, 1, 0, 2, 5,
       4, 3, 2, 4, 5, 2, 4, 2, 5, 5, 3, 2, 2, 2, 5, 4, 2, 2, 4, 4, 1, 2,
       4, 1, 1, 1, 2, 1, 1, 0, 3, 3, 2, 1, 1, 2, 2, 2, 2, 1, 4, 2, 1, 1,
       1, 0, 0, 0])
```

[190]:
```
x = pca_15_transform_df[:,0]
y = pca_15_transform_df[:,1]
sns.scatterplot(x = x,y = y,hue=labels,palette='Set1')

# fig = px.scatter(x=x, y=y, color=labels,height=500,width=500,labels={"x":"PC1","y":"PC2"})
# fig.show()
```

[190]:
```
<Axes: >
```

```
plt.ylabel('PC2')

# fig = px.scatter(x=x, y=y, color=labels,height=500,width=500,labels={"x":"PC1","y":"PC2"})
# fig.show()
```

[193]:
```
Text(0, 0.5, 'PC2')
```



From above plot,and analysis by using various feature selection and feature elimination techniques, we are able to reduce the curse of dimensionality and reduce the feature space from 203 features to just 15 features and still able to segment the dataset into 7 clusters/ classes using K Means

3. After applying principal component analysis (PCA) on a given dataset, it was found that the percentage of variance for the first N components is X%. How is this percentage of variance computed?

Percentage of variance is computed using the explained_variance_ratio parameter on the fitted pca object

The percentage of variance for the first 70 components (all components) is 100%
The percentage of variance for the first 16 components is:82.0%
The percentage of variance for the first 2 components is:42.0%

```
[51]: print(f"Precentage of variance explained by first 70 components:{round(pca.explained_variance_ratio_[:].sum(),2) * 100}%")
      print(f"Precentage of variance explained by first 16 components:{round(pca.explained_variance_ratio_[:16].sum(),2) * 100}%")
      print(f"Precentage of variance explained by first 2 components:{round(pca.explained_variance_ratio_[:3].sum(),2) * 100}%")

      Precentage of variance explained by first 70 components:100.0%
      Precentage of variance explained by first 16 components:82.0%
      Precentage of variance explained by first 2 components:42.0%

[ ]:
```

Part-2

Dataset description: This dataset includes data for the estimation of obesity levels in individuals based on their eating habits and physical condition. The data contains 17 attributes and 2111 records.

4. Create a machine learning (ML) model for predicting “weight” using all features except “NObeyesdad” and report observed performance. Explain your results based on following criteria:
a. What model have you selected for solving this problem and why?
b. Have you made any assumption for the target variable? If so, then why?
c. What have you done with text variables? Explain.
d. Have you optimized any model parameters? What is the benefit of this action?
e. Have you applied any steps for handling overfitting or underfitting issues? What is that?

a. What model have you selected for solving this problem and why?

In this Problem,Our aim is to predict the weight of the person based on certain predictor variables,this defines this as Regression problem and for this we will be applying various supervised regression algorithms and based on the performance of the model we will select the final model

We are using the following models :-
Linear Regression
Lasso Regression
Random forest Regressor
GradientBoostingRegressor
AdaBoostRegressor
XGBOOST Regressor

```
import matplotlib.pyplot as plt
```

[76]:
```
df = pd.read_csv('ObesityDataSet.csv')
df.head()
```

[76]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes

b. Have you made any assumption for the target variable? If so, then why?

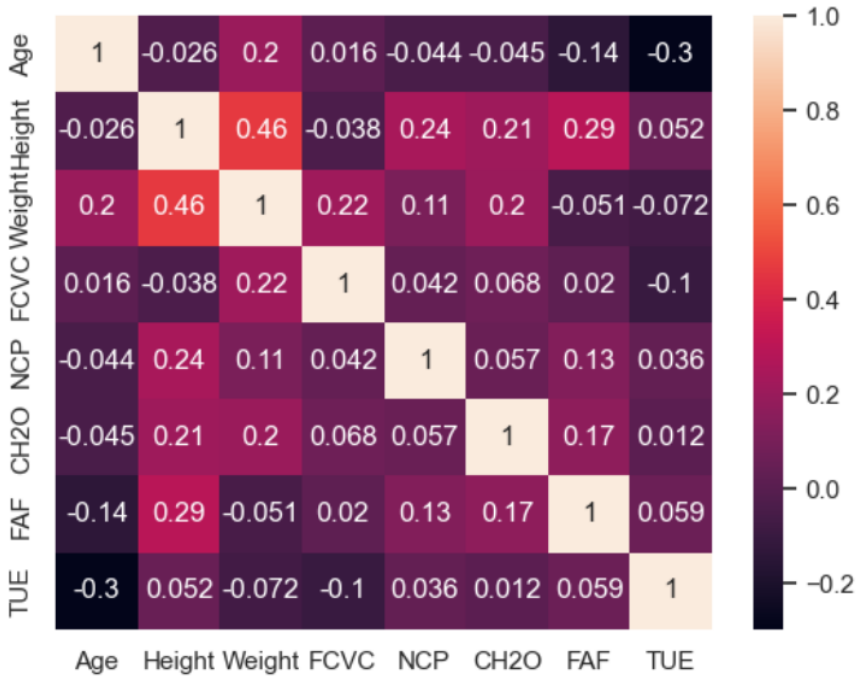
The Target variable is Weight and it is correlated with the predictor variables. So Linear regression is suitable to be applied on this first
Further assumption to be considered:--
The observations are independent of each other.
The errors follow a normal distribution.
The independent variables are not highly correlated with each other.
Homoscedasticity: The variance of the errors is constant across all levels of the independent variables

```
[82]: sns.scatterplot(y=df['Weight'],x = df['Height'],)
[82]: <Axes: xlabel='Height', ylabel='Weight'>
```

From above plot , It seems that Weight(Target) and Hieght(Predictor) are correlated to each other

Below Heatmap is created to check multicollinearity

```
[204]: sns.heatmap(df.corr(),annot = True)

[204]: <Axes: >
```



c. What have you done with text variables? Explain.

The columns which are not numerical will be converted to numeric columns using pd.get_dummies functions so that machine learning algorithm can treat those variables

Below categorical / text variables are converted into numeric

[91]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
dtypes: float64(8), object(8)
memory usage: 264.0+ KB
```

[93]:
```
print(f"Text variable columns :\n{list(df.select_dtypes(include=['object','category']).columns)}")
```

```
Text variable columns :
['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

[93]:
```
print(f"Text variable columns :\n{list(df.select_dtypes(include=['object','category']).columns)}")
```

```
Text variable columns :
['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

[94]:
```
df_converted = pd.get_dummies(df,drop_first=True)
df_converted.head()
```

[94]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	Gender_Male	family_history_with_overweight_yes	...	CAEC_no	SMOKE_yes	SCC_yes	CALC_Frequently	CALC
0	21.0	1.62	64.0	2.0	3.0	2.0	0.0	1.0	0	1	...	0	0	0	0	
1	21.0	1.52	56.0	3.0	3.0	3.0	3.0	0.0	0	1	...	0	1	1	0	
2	23.0	1.80	77.0	2.0	3.0	2.0	2.0	1.0	1	1	...	0	0	0	1	
3	27.0	1.80	87.0	3.0	3.0	2.0	2.0	0.0	1	0	...	0	0	0	1	
4	22.0	1.78	89.8	2.0	1.0	2.0	0.0	0.0	1	0	...	0	0	0	0	

5 rows × 23 columns

[95]:
```
df_converted.shape
```

[95]:
```
(2111, 23)
```

d. Have you optimized any model parameters? What is the benefit of this action?

Here , we are going to create various regression models with their parameters being tuned/optimized for better results.This would help in getting good result on testing data

[55]:
```
lr = LinearRegression()
model = create_model('Linear_Regression',lr,X_train_scaled,y_train,X_test_scaled,y_test)
```

```
Using Linear_Regression
Model score on training Data:0.5808885596803982
Model score on testing Data:0.5828135132679231
Mean_absolute_error: 13.653608738653872
Root_Mean_squared_error: 17.155531640918152
```

[56]:
```
ls = Lasso()
param = {'alpha':[0.1,0.5,1]}

skf = KFold(n_splits=5,shuffle=True)
grid = GridSearchCV(ls,param_grid=param,cv=skf,refit=True,verbose=1,return_train_score=True)
model = create_model('lasso',grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
result = pd.DataFrame(model.cv_results_).sort_values(by = 'rank_test_score')
result[['params','mean_test_score','mean_train_score']].head()
```

```
Using lasso
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Model score on training Data:0.5802176604982587
Model score on testing Data:0.5822939401233771
Mean_absolute_error: 13.710866378457993
Root_Mean_squared_error: 17.16621125333108
Best parameters for model:{'alpha': 0.1}
```

[56]:

	params	mean_test_score	mean_train_score
0	{'alpha': 0.1}	0.558384	0.581642
1	{'alpha': 0.5}	0.554537	0.574481
2	{'alpha': 1}	0.543102	0.560802

[57]:
```
el = ElasticNet()
param = {'alpha':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}

skf = KFold(n_splits=5,shuffle=True)

grid = GridSearchCV(el,param_grid=param,cv=skf,refit=True,verbose=1,return_train_score=True)
model = create_model("ElasticNet",grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
result = pd.DataFrame(model.cv_results_).sort_values(by = 'rank_test_score')
result[['params','mean_test_score','mean_train_score']].head()
```

```
Using ElasticNet
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Model score on training Data:0.5784871951019938
Model score on testing Data:0.5806723262284612
Mean_absolute_error: 13.76348204557237
Root_Mean_squared_error: 17.199500214860194
Best parameters for model:{'alpha': 0.1}
```

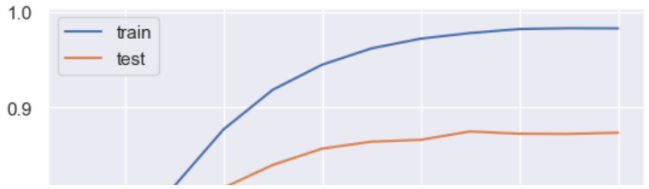
[57]:

	params	mean_test_score	mean_train_score
0	{'alpha': 0.1}	0.565800	0.579951
1	{'alpha': 0.2}	0.562454	0.574946
2	{'alpha': 0.3}	0.557300	0.568649
3	{'alpha': 0.4}	0.551693	0.561954

4	{'alpha': 0.5}	0.545924	0.555166

[58]:
```
rf = RandomForestRegressor()

test = []
train = []
r = [3,4,5,6,7,8,9,10,11,12,13,14]
for t in r:
    rf = RandomForestRegressor(max_depth=t)
    rf.fit(X_train_scaled,y_train)
    y_pred = rf.predict(X_test_scaled)
    train.append(rf.score(X_train_scaled,y_train))
    test.append(rf.score(X_test_scaled,y_test))

plt.plot(r,train,label='train')
plt.plot(r,test,label='test')
plt.ylabel('score')
plt.xlabel('depth of tree')
plt.legend()
```

[58]:
```
<matplotlib.legend.Legend at 0x2b9276878b0>
```



[59]:
```
param = {'n_estimators':[100,200,250,300],
        'max_features' : ["auto", "sqrt", "log2"],
        'max_depth':[5,6,7,8,9]}

skf = KFold(n_splits=5,shuffle=True)

grid = GridSearchCV(rf,param_grid=param,cv=skf,refit=True,verbose=1,return_train_score=True)
model = create_model("Random_forest",grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
result = pd.DataFrame(model.cv_results_).sort_values(by = 'rank_test_score')
result[['params','mean_test_score','mean_train_score']].head()
```

```
Using Random_forest
Fitting 5 folds for each of 60 candidates, totalling 300 fits
Model score on training Data:0.9614081298279076
Model score on testing Data:0.8663450417519792
Mean_absolute_error: 6.213822864175021
Root_Mean_squared_error: 9.710276624135686
Best parameters for model:{'max_depth': 9, 'max_features': 'auto', 'n_estimators': 100}
```

[59]:

	params	mean_test_score	mean_train_score
48	{'max_depth': 9, 'max_features': 'auto', 'n_es...	0.859221	0.963668
51	{'max_depth': 9, 'max_features': 'auto', 'n_es...	0.859104	0.963896
50	{'max_depth': 9, 'max_features': 'auto', 'n_es...	0.857884	0.963792
57	{'max_depth': 9, 'max_features': 'log2', 'n_es...	0.856880	0.930614
49	{'max_depth': 9, 'max_features': 'auto', 'n_es...	0.856354	0.963730

49	{'max_depth': 9, 'max_features': 'auto', 'n_es...	0.856354	0.963730

[ ]:

[60]:
```
xg = XGBRegressor()
param = {'n_estimators':[100,200,250],
        # 'max_features' : ["auto", "sqrt", "log2"],
         'learning_rate':[0.1,0.05,0.2],
        'max_depth':[5,6,7,8,9]}

skf = KFold(n_splits=5,shuffle=True)

grid = GridSearchCV(xg,param_grid=param,cv=skf,refit=True,verbose=1,return_train_score=True)
model = create_model('XGBoost',grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
result = pd.DataFrame(model.cv_results_).sort_values(by = 'rank_test_score')
result[['params','mean_test_score','mean_train_score']].head()
```

```
Using XGBoost
Fitting 5 folds for each of 45 candidates, totalling 225 fits
Model score on training Data:0.9981127252125205
Model score on testing Data:0.8725395224306163
Mean_absolute_error: 5.808542445728975
Root_Mean_squared_error: 9.48258699213045
Best parameters for model:{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200}
```

[60]:

	params	mean_test_score	mean_train_score
7	{'learning_rate': 0.1, 'max_depth': 7, 'n_esti...	0.890363	0.998714
8	{'learning_rate': 0.1, 'max_depth': 7, 'n_esti...	0.890311	0.999360

7	{'learning_rate': 0.1, 'max_depth': 7, 'n_esti...	0.890363	0.998714
8	{'learning_rate': 0.1, 'max_depth': 7, 'n_esti...	0.890311	0.999360
26	{'learning_rate': 0.05, 'max_depth': 8, 'n_est...	0.890064	0.998198
25	{'learning_rate': 0.05, 'max_depth': 8, 'n_est...	0.889516	0.996847
10	{'learning_rate': 0.1, 'max_depth': 8, 'n_esti...	0.888763	0.999701

[61]:
```
gd = GradientBoostingRegressor()
param = {'n_estimators':[100,200,250,300,400],
         'learning_rate':[0.1,0.05,0.2],
        'max_depth':[5,6,7,8,9]}
        # 'max_depth':[3,4,5]}

skf = KFold(n_splits=5,shuffle=True)

grid = GridSearchCV(gd,param_grid=param,cv=skf,refit=True,verbose=1,return_train_score=True)
model = create_model('GradientBoostingRegressor',grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
result = pd.DataFrame(model.cv_results_).sort_values(by = 'rank_test_score')
result[['params','mean_test_score','mean_train_score']].head()
```

```
Using GradientBoostingRegressor
Fitting 5 folds for each of 75 candidates, totalling 375 fits
Model score on training Data:0.9996550407663248
Model score on testing Data:0.8683131457166835
Mean_absolute_error: 5.656004997996309
Root_Mean_squared_error: 9.638518307154278
Best parameters for model:{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 250}
```

After performing various optimization techniques we get the data frame which shows the performance of various models being used.

22	{'learning_rate': 1.0, 'n_estimators': 200}	0.695172	0.730378
18	{'learning_rate': 0.1, 'n_estimators': 300}	0.692304	0.730851
21	{'learning_rate': 1.0, 'n_estimators': 100}	0.691165	0.725933

[63]:
```
df = pd.DataFrame({'Name':Name,'Training_Score':training_score,'Testing_score':testing_score,
                   'Mean_absolute_error':Mean_absolute_error,
                  'Root_Mean_absolute_error':Root_Mean_absolute_error})
df.sort_values(by = ['Testing_score'],ascending=False)
```

[63]:

	Name	Training_Score	Testing_score	Mean_absolute_error	Root_Mean_absolute_error
4	XGBoost	0.998113	0.872540	5.808542	9.482587
5	GradientBoostingRegressor	0.999655	0.868313	5.656005	9.638518
3	Random_forest	0.961408	0.866345	6.213823	9.710277
6	AdaBoosting	0.722816	0.680484	12.624360	15.013614
0	Linear_Regression	0.580889	0.582814	13.653609	17.155532
1	lasso	0.580218	0.582294	13.710866	17.166211
2	ElasticNet	0.578487	0.580672	13.763482	17.199500

e. Have you applied any steps for handling overfitting or underfitting issues? What is that?

In order to handle overfitting/underfitting in developing the model, various techniques are performed to tackle this issue:-

1. Regularisation- is a technique to prevent the model from overfitting by adding extra information to it.
In the regularization technique, we reduce the magnitude of the features by keeping the same number of features."
Ridge regression is one of regularization technique of linear regression in which a small amount of bias is introduced
It is a regularization technique, which is used to reduce the complexity of the model. It is also called L2 regularization.
In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called Ridge Regression penalty
Lasso regression is another regularization technique to reduce the complexity of the model.
It stands for Least Absolute and Selection Operator.
It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
It is also called L1 regularization.

Lasso Regression is used above along with ElasticNet with Linear Regression

2. Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also

say that it is a technique to check how a statistical model generalizes to an independent dataset.
K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called folds. For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set. This approach is a very popular CV approach because it is easy to understand, and the output is less biased than other methods.

This technique is used in all above trained models

3. Ensemble learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.
Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly
Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree
Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree

All above three ensemble learning techniques were used in the above model training

4. Hyperparameter Tuning -there are parameters that are internally learned from the given dataset and derived from the dataset, they are represented in making predictions, classification and etc., These are so-called Model Parameters, and they are varying with respect to the nature of the data we couldn't control this since it depends on the data.
Grid-Search: To implement the Grid-Search, we have a Scikit-Learn library called GridSearchCV. The computational time would be long, but it would reduce the manual efforts by avoiding the 'n' number of lines of code. Library itself performs the search operations and returns the performing model and its score. In which each model are built for each permutation of a given hyperparameter, internally it would be evaluated and ranked across the given cross-validation folds
Grid Search Technique is also used in above models to have optimal hyperparameters

[64]:
```
df = pd.DataFrame({'Name':Name,'Training_Score':training_score,'Testing_score':testing_score,
                   'Mean_absolute_error':Mean_absolute_error,
                  'Root_Mean_absolute_error':Root_Mean_absolute_error})
df.sort_values(by = ['Testing_score'],ascending=False)
```

[64]:

	Name	Training_Score	Testing_score	Mean_absolute_error	Root_Mean_absolute_error
4	XGBoost	0.999875	0.879702	5.491460	9.212307
3	Random_forest	0.962086	0.865006	6.232020	9.758789
5	GradientBoostingRegressor	0.999961	0.861243	5.747914	9.893880
6	Ada	0.729617	0.691378	12.392426	14.755445
0	Linear_Regression	0.580889	0.582814	13.653609	17.155532
1	lasso	0.580218	0.582294	13.710866	17.166211
2	ElasticNet	0.578487	0.580672	13.763482	17.199500

From above dataframe, we can see :-
Linear regression and LASSO performed poorly
ADABoost regressor did a moderate job
XGBOOST, GradientBoosting Regressor and Random Forest performed decent
But XGBOOST and Gradient Boosting overfits the data as they have almost 100% training score
Random Forest can be chosen as the final model for this as it is performing well both on training as well as on testing data

ADABoost regressor did a moderate job
XGBOOST, GradientBoosting Regressor and Random Forest performed decent But XGBOOST and Gradient Boosting overfits the data as they have almost 100% training score
Random Forest can be chosen as the final model for this as it is performing well both on training as well as on testing data

[65]:
```
rf = RandomForestRegressor(random_state=42)
param = {'n_estimators':[50,60,70,80,90,100],
         'min_samples_leaf':[1,2,3,4],
         'min_samples_split':[2,3,4],
        'max_features' : ["auto", "sqrt", "log2"],
        'max_depth':[3,4,5,6,7,8,9]}

skf = KFold(n_splits=5,shuffle=True)
grid = GridSearchCV(rf,param_grid=param,cv=skf,refit=True,verbose=1)
model = create_model("Random_forest",grid,X_train_scaled,y_train,X_test_scaled,y_test)
print(f"Best parameters for model:{model.best_params_}")
```

```
Using Random_forest
Fitting 5 folds for each of 1512 candidates, totalling 7560 fits
Model score on training Data:0.9605529166947031
Model score on testing Data:0.8645090332782871
Mean_absolute_error: 6.340485191965618
Root_Mean_squared_error: 9.776743822014172
Best parameters for model:{'max_depth': 9, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimator
s': 90}
```

Random forest model was further optimized to on various parameters such as n_estimators,max_depth,max_features etc to handle overfitting as well as underfitting
The model is now showing decent score on training as well as on testing data

References

https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/#5
https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/ https://hastie.su.domains/ISLP/ISLP_website.pdf
https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
https://www.bioinformatics.babraham.ac.uk/training/10XRNASeq/Dimension%20Reduction.pdf
http://theprofessionalspoint.blogspot.com/2019/03/advantages-and-disadvantages-of-t-sne.html
https://www.educba.com/density-based-clustering/
https://plotly.com/graphing-libraries/
https://seaborn.pydata.org/index.html
https://pypi.org/project/feature-engine/
https://www.datacamp.com/blog