Project 2

Group 26

Members:

1.Durgalakshmi Sundaraman(dxs190016)
2.Kruthika Natarajan Nirmala(kxn190008)

Regression Task:

- Apply any two models with bagging and any two models with pasting.
- Apply any two models with adaboost boosting
- Apply one model with gradient boosting
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA.
- Apply deep learning models covered in class

DataSet

This dataset is taken from kaggle. It contains information of the living and health standards of people in various countries in this is **life expectancy** variable . The model works to predict the life expectancy in years of an individual
The dataset has **1535** observations with **22** columns

Importing necessary libraries for the project

In [1]:
```
#import neccessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from math import sqrt
import warnings
warnings.filterwarnings("ignore")
np.random.seed(42)
```

Importing the Dataset for classification

In [2]:
```
#reading the dataset
df_life = pd.read_excel("C:\\Users\\sundh\\Desktop\\ML\\ML_Project1_Group26\\ML Project\\ML Regression
\\Project1_Regression_Data.xlsx")
df_life.shape
```

Out[2]:
```
(1535, 22)
```

Column : Description
Country : Country the respondent is in
Year : Year of the response
Status : Status of the country
Life expectancy : Average life expectancy as age
Adult Mortality : Average Adult mortality rate
infant deaths : Average infant deaths
Alcohol : Alcohol consumption
percentage expenditure : Average medicalExpenditure
Hepatitis B : Hepatitis B immunisation among infants
Measles : No. measles cases reported
BMI : BMI of population
under-five deaths :Number of under five deaths
Polio : Polio coverage among
Diphtheria : Diphtheria immunisation
HIV/AIDS : HIV deaths per 1000
GDP : GDP of the country
Population Population of the country
thinness 1-19 years : prevelance of thinness in 1-19 year age group
thinness 5-9 years : prevelance of thinness in 5-9 year age group
Income composition of resources : Income composition of population
Schooling : Number of years of schooling

Exploratory Data Analysis

The exploratory data analysis is performed on the dataset to understand the variations, range, detect outliers and collinearity if it exists in the dataset.

Check for null

Upon checking for null we find that 1385 of the data cells in the dataset have a randomly disbursed null presence

In [3]:
```
#check the datatypes
df_life.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1535 entries, 0 to 1534
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Country                          1534 non-null   object
 1   Year                             1534 non-null   float64
 2   Status                           1533 non-null   object
 3   Life expectancy                  1527 non-null   float64
 4   Adult Mortality                  1524 non-null   float64
 5   infant deaths                    1534 non-null   float64
 6   Alcohol                          1433 non-null   float64
 7   percentage expenditure           1533 non-null   float64
 8   Hepatitis B                      1229 non-null   float64
 9   Measles                          1535 non-null   int64
 10  BMI                              1514 non-null   float64
 11  under-five deaths                1535 non-null   int64
 12  Polio                            1521 non-null   float64
 13  Total expenditure                1415 non-null   float64
 14  Diphtheria                       1523 non-null   float64
 15   HIV/AIDS                        1533 non-null   float64
 16  GDP                              1304 non-null   float64
 17  Population                       1197 non-null   float64
 18  thinness  1-19 years             1515 non-null   float64
 19  thinness 5-9 years               1515 non-null   float64
 20  Income composition of resources  1448 non-null   float64
 21  Schooling                        1449 non-null   float64
dtypes: float64(18), int64(2), object(2)
memory usage: 264.0+ KB
```

In [4]:
```
#sum of the total nulls in dataset
df_life.isna().sum().sum()
```

Out[4]:
```
1385
```

In [5]:
```
#stripping blanks from column headers
df_life.columns = df_life.columns[:].str.strip() #strip removes both start and end
df_life.columns
```

Out[5]:
```
Index(['Country', 'Year', 'Status', 'Life expectancy', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles', 'BMI', 'under-five deaths', 'Polio', 'Total expenditure',
       'Diphtheria', 'HIV/AIDS', 'GDP', 'Population', 'thinness  1-19 years',
       'thinness 5-9 years', 'Income composition of resources', 'Schooling'],
      dtype='object')
```

Dropping unwanted columns

The column country is dropped here as it doesnt add to our data prediction

In [6]:
```
#droping column country
df_life.drop(['Country'],inplace=True,axis=1)
```

Handling Null

To handle the null we check the data in the first five rows

In [7]:
```
#reading the first five rows in the dataset
df_life.head()
```

Out[7]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Polio	Total expenditure	Diphtheria
0	2011.0	Developed	73.7	144.0	1.0	10.67	875.149519	96.0	157	63.3	...	95.0	6.88	95.0
1	2008.0	Developing	61.4	217.0	8.0	0.02	61.762638	74.0	4	26.1	...	73.0	3.23	74.0
2	2007.0	Developing	73.4	165.0	9.0	8.70	0.000000	62.0	32	57.6	...	67.0	5.25	62.0
3	2013.0	Developing	74.3	89.0	8.0	0.09	2047.113102	98.0	1164	66.4	...	NaN	4.25	98.0
4	2011.0	Developed	74.3	139.0	2.0	9.10	10.325889	96.0	4189	57.7	...	89.0	5.53	89.0

5 rows × 21 columns

In [8]:
```
#sum of null by column
df_life.isna().sum()
```

Out[8]:
```
Year                                 1
Status                               2
Life expectancy                      8
Adult Mortality                     11
infant deaths                        1
Alcohol                            102
percentage expenditure               2
Hepatitis B                        306
Measles                              0
BMI                                 21
under-five deaths                    0
Polio                               14
Total expenditure                  120
Diphtheria                          12
HIV/AIDS                             2
GDP                                231
Population                         338
thinness  1-19 years                20
thinness 5-9 years                  20
Income composition of resources     87
Schooling                           86
dtype: int64
```

Handle Null To handle the null, we choose to impute the mean value of the numeric columns and for categorical variables we have imputed the value with most count

In [9]:
```
# Replacing null values with mean of the column values
df_life['Life expectancy'].fillna(df_life['Life expectancy'].mean(), inplace=True)
df_life['Adult Mortality'].fillna(df_life['Adult Mortality'].mean(), inplace=True)
df_life['Alcohol'].fillna(df_life['Alcohol'].mean(), inplace=True)
df_life['Hepatitis B'].fillna(df_life['Hepatitis B'].mean(), inplace=True)
df_life['BMI'].fillna(df_life['BMI'].mean(), inplace=True)
df_life['Polio'].fillna(df_life['Polio'].mean(), inplace=True)
df_life['Total expenditure'].fillna(df_life['Total expenditure'].mean(), inplace=True)
df_life['Diphtheria'].fillna(df_life['Diphtheria'].mean(), inplace=True)
df_life['GDP'].fillna(df_life['GDP'].mean(), inplace=True)
df_life['Population'].fillna(df_life['Population'].mean(), inplace=True)
df_life['infant deaths'].fillna(df_life['infant deaths'].mode()[0], inplace=True)
df_life['percentage expenditure'].fillna(df_life['percentage expenditure'].mean(), inplace=True)
df_life['HIV/AIDS'].fillna(df_life['BMI'].mean(), inplace=True)

df_life['thinness  1-19 years'].fillna(df_life['thinness  1-19 years'].mean(), inplace=True)
df_life['thinness 5-9 years'].fillna(df_life['thinness 5-9 years'].mean(), inplace=True)
df_life['Income composition of resources'].fillna(df_life['Income composition of resources'].mean(), in
place=True)
df_life['Schooling'].fillna(df_life['Schooling'].mean(), inplace=True)
df_life['Year'].fillna(df_life['Year'].value_counts().index[0], inplace=True)
df_life['Status'].fillna(df_life['Status'].value_counts().index[0], inplace=True)
```

In [10]:
```
#checking for nulls in each column
df_life.isna().sum()
```

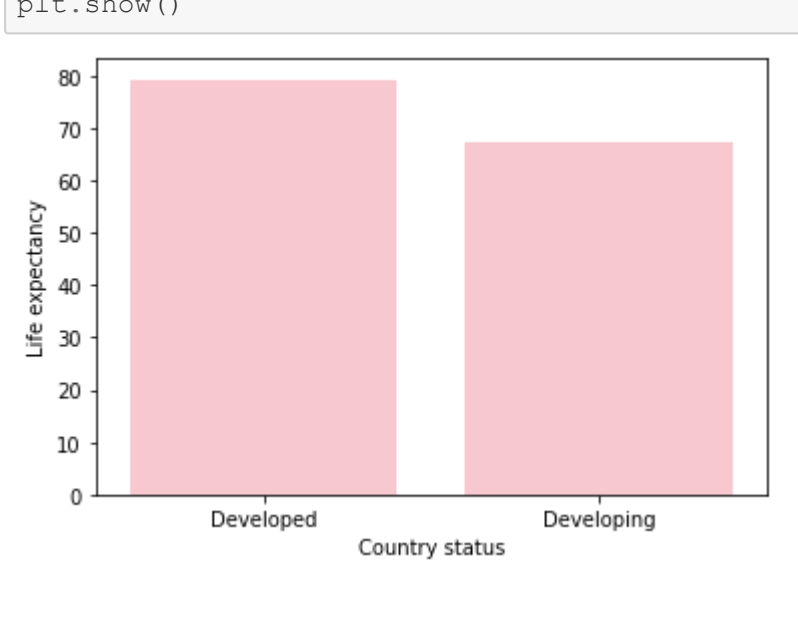
Out[10]:
```
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
BMI                                0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
HIV/AIDS                           0
GDP                                0
Population                         0
thinness  1-19 years               0
thinness 5-9 years                 0
Income composition of resources    0
Schooling                          0
dtype: int64
```

Data Visualisation

We check the spread of the target variable Life expectancy.

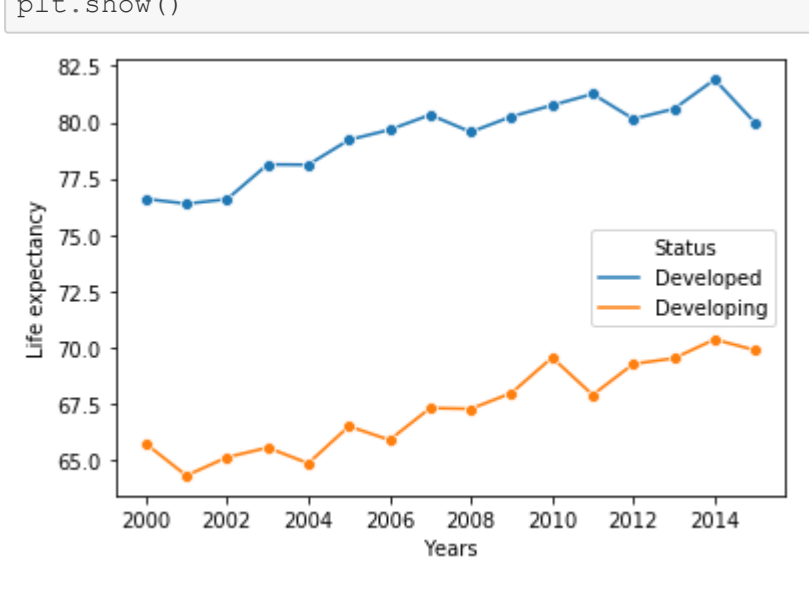
The developed countries more life expectancy than developing countries

In [11]:
```
#life expectancy in Developed and Developing countries
#plt.figure(figsize=(16, 6))
plot2=sns.barplot(data=df_life,x='Status',y='Life expectancy',color='Pink',ci=None)
plot2.set(xlabel="Country status", ylabel="Life expectancy")
plt.show()
```



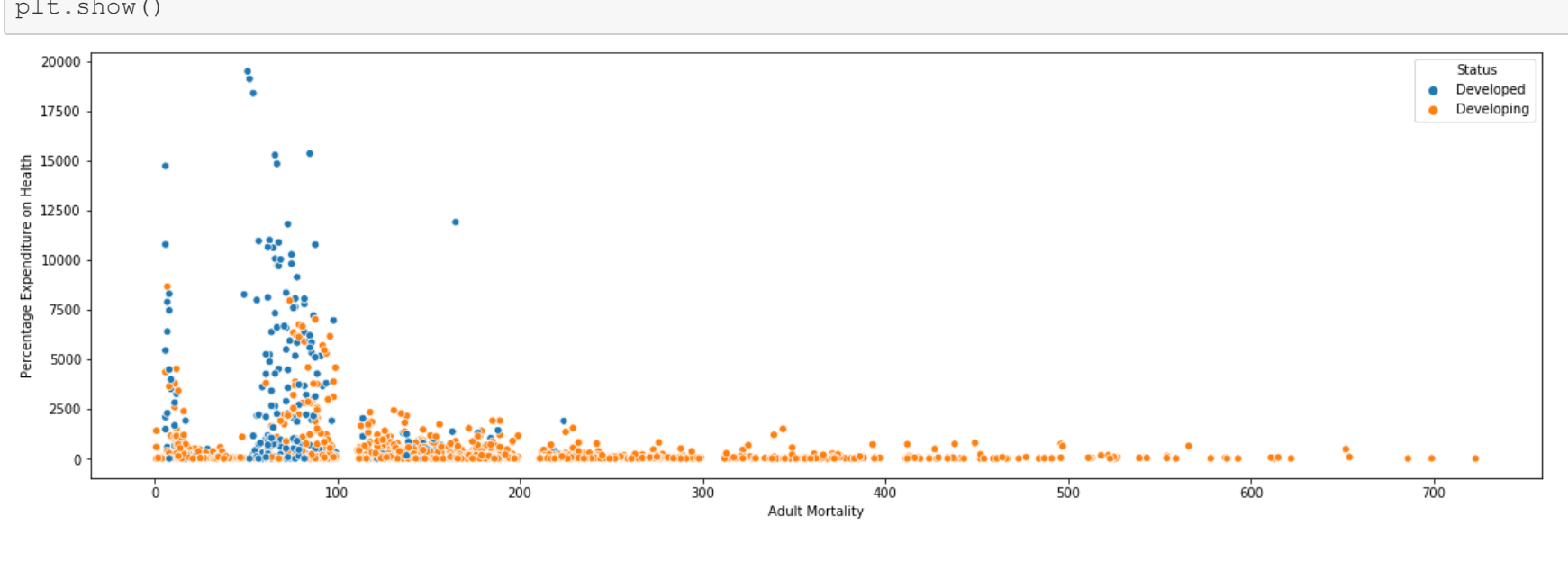
Over years the life expectancy has been gradually growing

In [12]:
```
#life expectancy over the years
#plt.figure(figsize=(16, 6))
plot2=sns.lineplot(data=df_life,x='Year',y='Life expectancy',hue='Status',marker='o',ci=None)
plot2.set(xlabel="Years", ylabel="Life expectancy")
plt.show()
```



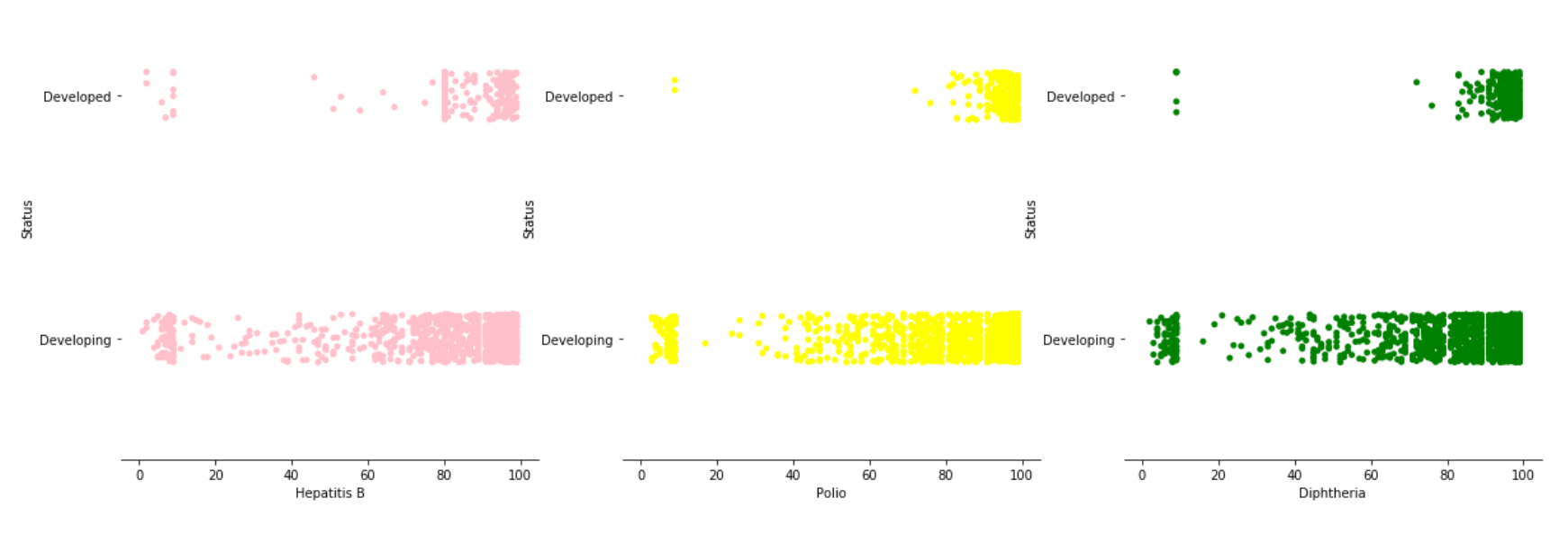
Mortality rates of the developed countries are lesser while in developing countries its dispersed The expenditure on fealth is more for developed countries

In [13]:
```
#PLotting the scatterplot to depict relationship between Adult Mortality and Percentage Expenditure on
 Health
plt.figure(figsize=(20, 6))
plot2=sns.scatterplot(data=df_life,x='Adult Mortality',y='percentage expenditure',hue='Status',color='P
ink')#,order=[0,1]
plot2.set(xlabel="Adult Mortality", ylabel="Percentage Expenditure on Health")
plt.show()
```



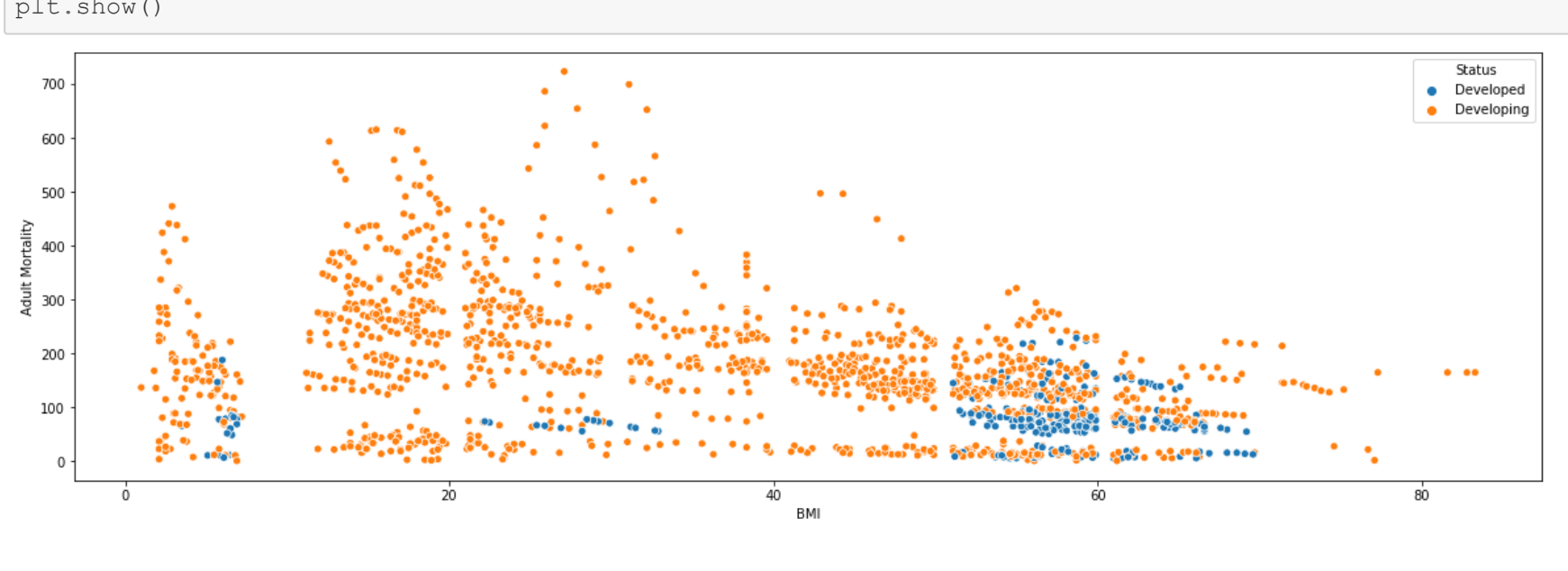
Immunisations for various diseases in both status of countries. We can see that the developed countries are more immunised

In [14]:
```
#Plotting the number of different immunizations given in developing and developed countries
fig, ax =plt.subplots(1,3,figsize=(20, 7),sharex=True)
sns.despine(left=True)
sns.stripplot(data=df_life,x='Hepatitis B',y='Status', ax=ax[0],color='Pink')
sns.stripplot(data=df_life,x='Polio',y='Status', ax=ax[1],color='Yellow')
sns.stripplot(data=df_life,x='Diphtheria',y='Status',ax=ax[2],color='Green')
plt.show()
```



Mortality rates are spread across all BMI counts for developing countries

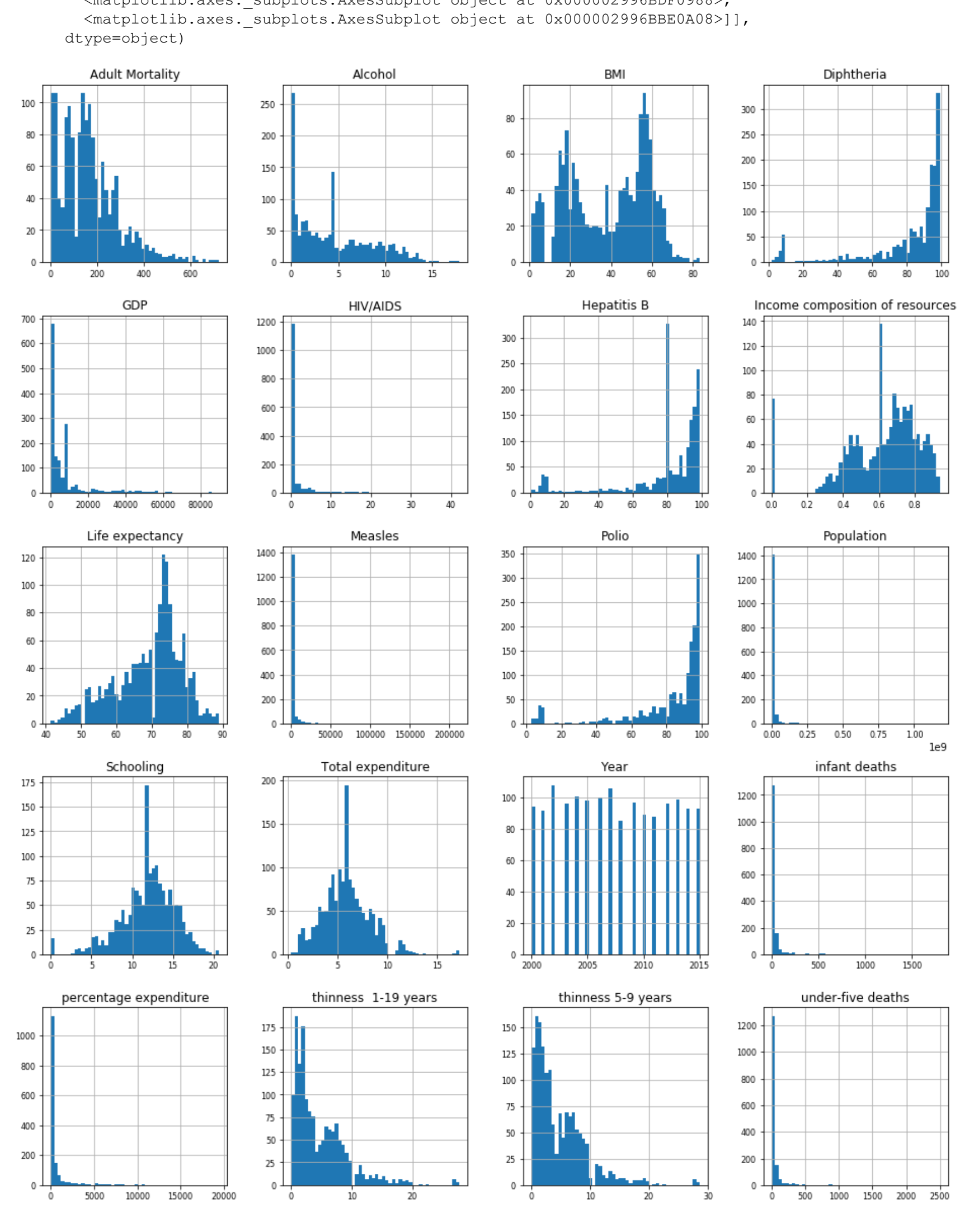
In [15]:
```
#PLotting the scatterplot of BMI and Adult Mortality
plt.figure(figsize=(20, 6))
plot2=sns.scatterplot(data=df_life,x='BMI',y='Adult Mortality',hue='Status',color='Pink')#,order=[0,1]
plot2.set(xlabel="BMI", ylabel="Adult Mortality")
plt.show()
```



Distribution of various variables

In [16]:
```
#Further exploration using histogram to check the distributions - EDA
df_life.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8)
```

Out[16]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002996C71F7C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996C806808>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996C83F088>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996C875B48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002996CADEC88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CB16D08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CB4EE08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CB87F48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002996CB91B48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CBCAD08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CC36248>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CC71348>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002996CCA8448>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CCE0588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CD1DE88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996CD52748>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002996C60A648>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996C5AF248>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996BDF0988>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002996BBE0A08>]],
      dtype=object)
```



Handle categorical variables

The categorical variables need to be converted to dummy variables which add more columns. Categorical variables will be represented as 0 or 1 in the columns

In [17]:
```
# creating dummy variables for the Status variable
cols = pd.get_dummies(df_life['Status'], prefix= 'Status')
df_life[cols.columns] = cols
df_life.drop('Status', axis = 1, inplace = True)
```

In [18]:
```
# creating dummy variables for the Year variable
cols = pd.get_dummies(df_life['Year'], prefix= 'Year')
df_life[cols.columns] = cols
df_life.drop('Year', axis = 1, inplace = True)
```

In [19]:
```
#the shape of the df now changes because of addition of dummy variables
df_life.shape
```

Out[19]:
```
(1535, 37)
```

In [20]:
```
#Checking the final list of columns
for col in df_life.columns:
    print(col)
```

```
Life expectancy
Adult Mortality
infant deaths
Alcohol
percentage expenditure
Hepatitis B
Measles
BMI
under-five deaths
Polio
Total expenditure
Diphtheria
HIV/AIDS
GDP
Population
thinness  1-19 years
thinness 5-9 years
Income composition of resources
Schooling
Status_Developed
Status_Developing
Year_2000.0
Year_2001.0
Year_2002.0
Year_2003.0
Year_2004.0
Year_2005.0
Year_2006.0
Year_2007.0
Year_2008.0
Year_2009.0
Year_2010.0
Year_2011.0
Year_2012.0
Year_2013.0
Year_2014.0
Year_2015.0
```

Train and Test Split

The test and train split is done with default of 0.25

In [21]:
```
#assigning life expectancy as target column
X = df_life.drop('Life expectancy',axis=1).values
y=df_life['Life expectancy'].values
```

In [22]:
```
#just checking the predictor variables
X
```

Out[22]:
```
array([[1.440e+02, 1.000e+00, 1.067e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [2.170e+02, 8.000e+00, 2.000e-02, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.650e+02, 9.000e+00, 8.700e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       ...,
       [7.300e+01, 0.000e+00, 7.530e+00, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       [2.500e+01, 1.000e+00, 5.800e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.400e+01, 1.000e+00, 4.170e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00]])
```

In [23]:
```
#checking the target variable
y
```

Out[23]:
```
array([73.7, 61.4, 73.4, ..., 88. , 66.3, 78. ])
```

In [24]:
```
#creating the formal X and y split
X_train_org,X_test_org,y_train,y_test= train_test_split(X,y, random_state=0)
```

In [25]:
```
df_life.isna().sum()
```

Out[25]:
```
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
BMI                                0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
HIV/AIDS                           0
GDP                                0
Population                         0
thinness  1-19 years               0
thinness 5-9 years                 0
Income composition of resources    0
Schooling                          0
Status_Developed                   0
Status_Developing                  0
Year_2000.0                        0
Year_2001.0                        0
Year_2002.0                        0
Year_2003.0                        0
Year_2004.0                        0
Year_2005.0                        0
Year_2006.0                        0
Year_2007.0                        0
Year_2008.0                        0
Year_2009.0                        0
Year_2010.0                        0
Year_2011.0                        0
Year_2012.0                        0
Year_2013.0                        0
Year_2014.0                        0
Year_2015.0                        0
dtype: int64
```

In [26]:
```
df_life.shape
```

Out[26]:
```
(1535, 37)
```

Scaling

Scaling is done with MinMax scaler which scales the data with respect to the minimum and maximum values, this is better than the standard scaler which will mostly incline towards the mean of the data.

In [27]:
```
#scaling and transform of X_train and X_test
scaler= MinMaxScaler()
scaler.fit(X_train_org)
X_train= scaler.transform(X_train_org)
X_test= scaler.transform(X_test_org)


#Scores
train_score=[]
test_score=[]
```

In [28]:
```
reg_model_para = pd.DataFrame(columns=["Regressor", "Avg Train Score", "Avg Test Score", "Best Hyperpar
ameters"])
```

BAGGING (ridge and lasso)

Bagging is an aggregation ensemble meta-algorithm designed to improve the stability and accuracy of algorithms. It helps to reduce overfitting. One main parameter passed is the bootstrap parameter.It uses sampling with replacement.

N_estimators and max_samples are parameters that control the number of decision trees and sample size used.

We have use bagging on ridge and lasso models.

Bagging -- Ridge

Running a grid search for the best parameters for bagging

In [29]:
```
# Grid search to find the best parameter for bagging
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
param_grid = {'n_estimators': [50,100,200,400,500],
              'max_samples': [50,100,200,400,500]}

grid_search = GridSearchCV(BaggingRegressor(), param_grid, cv=10, return_train_score=True)
grid_search.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_samples': 500, 'n_estimators': 500}
Best cross-validation score: 0.93
```

The best parameters derived are max_samples: 500 and n_estimators: 500

In [30]:
```
train_score_list = []
test_score_list = []
ridge = Ridge(alpha=0.01)
bag_reg = BaggingRegressor(ridge, n_estimators=500, max_samples=500, bootstrap=True, n_jobs=-1, random_
state=0)

bag_reg.fit(X_train,y_train)
train_score_list.append(bag_reg.score(X_train,y_train))
test_score_list.append(bag_reg.score(X_test, y_test))
y_preds = bag_reg.predict(X_test)

print("Train score obtained in ridge after bagging:",train_score_list)
print("Test score obtained in ridge after bagging::",test_score_list)
```

```
Train score obtained in ridge after bagging: [0.8104902486858067]
Test score obtained in ridge after bagging:: [0.8005879464653182]
```

Bagging -- Lasso

Running a grid search for the best parameters for bagging

The best parameters derived are max_samples: 500 and n_estimators: 500

In [31]:
```
train_score_list = []
test_score_list = []


lasso = Lasso(alpha=0.01)
bag_reg = BaggingRegressor(lasso, n_estimators=500, max_samples=500, bootstrap=True, n_jobs=-1, random_
state=0)

bag_reg.fit(X_train,y_train)
train_score_list.append(bag_reg.score(X_train,y_train))
test_score_list.append(bag_reg.score(X_test, y_test))
y_preds = bag_reg.predict(X_test)


print("Train score obtained in lasso after bagging:",train_score_list)
print("Test score obtained in lasso after bagging:",test_score_list)
```

```
Train score obtained in lasso after bagging: [0.8053297218331958]
Test score obtained in lasso after bagging: [0.8041606386839412]
```

PASTING

Pasting is an aggregation ensemble meta-algorithm designed to improve the stability and accuracy of algorithms. It helps to reduce overfitting. One main parameter passed is the bootstrap parameter which is set to false.It uses sampling without replacement.

N_estimators and max_samples are parameters that control the number of decision trees and sample size used.

Pasting is applied on Ridge and Lasso models

Pasting -- Ridge

In [32]:
```
# Grid search to find the best parameter for pasting
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
param_grid = {'n_estimators': [50,100,200,500],
              'max_samples': [50,100,200,400,500]}

grid_search = GridSearchCV(BaggingRegressor(), param_grid, cv=5, return_train_score=True)
grid_search.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_samples': 500, 'n_estimators': 500}
Best cross-validation score: 0.93
```

The best parameters derived are max_samples: 500 and n_estimators: 500

In [33]:
```
train_score_list = []
test_score_list = []

ridge = Ridge(alpha=0.01)
bag_reg = BaggingRegressor(ridge, n_estimators=500, max_samples=500, bootstrap=False, n_jobs=-1, random
_state=0)
bag_reg.fit(X_train, y_train)
train_score_list.append(bag_reg.score(X_train, y_train))
test_score_list.append(bag_reg.score(X_test, y_test))

y_preds = bag_reg.predict(X_test)

print("Train score obtained in lasso after pasting:",train_score_list)
print("Test score obtained in lasso after pasting:",test_score_list)
```

```
Train score obtained in lasso after pasting: [0.8111263573469497]
Test score obtained in lasso after pasting: [0.8025778505057833]
```

Pasting -- Lasso

The best parameters derived for pasting as max_samples: 500 and n_estimators: 500

In [34]:
```
train_score_list = []
test_score_list = []
lasso = Lasso(alpha=0.01)
bag_reg = BaggingRegressor(lasso, n_estimators=500, max_samples=500, bootstrap=False, n_jobs=-1, random
_state=0)
bag_reg.fit(X_train,y_train)
train_score_list.append(bag_reg.score(X_train,y_train))
test_score_list.append(bag_reg.score(X_test, y_test))
y_preds = bag_reg.predict(X_test)

print("Train score obtained in lasso after pasting:",train_score_list)
print("Test score obtained in lasso after pasting:",test_score_list)
```

```
Train score obtained in lasso after pasting: [0.8051845689742916]
Test score obtained in lasso after pasting: [0.8050358150909892]
```

ADABOOST (KNN and RIDGE)

Adaboost turns multiple weak classifiers into a strong classifier. The important parameters used are n_estimators and learning_rate.

Learning rate shrinks the contribution of each classifier by learning_rate chosen.

Adaboost are used on models KNN and Ridge

Adaboost -- KNN

The best parameters for Adaboost are identified with gridsearch

In [35]:
```
from sklearn.ensemble import AdaBoostRegressor
parameters = {'n_estimators': [50,100,200,500],
              'learning_rate': [.01, .05, .1, 1]}

grid_search = GridSearchCV(AdaBoostRegressor(random_state = 0), param_grid = parameters, scoring='neg_mean_squared_error', cv=5, n_jobs = -1)

grid_search = grid_search.fit(X_train,y_train)

print("Grid search best score:", grid_search.best_score_)
print("Grid search best parameters:", grid_search.best_params_)
```

```
Grid search best score: -11.061216406591571
Grid search best parameters: {'learning_rate': 1, 'n_estimators': 200}
```

The best parameters are n_estimators and learning rate are used on KNN with neighbours-4 as idenfied before

In [36]:
```
from sklearn.ensemble import AdaBoostRegressor
ada_reg = AdaBoostRegressor(KNeighborsRegressor(n_neighbors=4), n_estimators=200,  learning_rate=1, random_state=0)
ada_reg.fit(X_train, y_train)
y_pred = ada_reg.predict(X_test)
print('KNN regressor score on Train Set after Adaboost Boosting: {:.3f}'.format(ada_reg.score(X_train, y_train)))
print('KNN regressor score on Test Set after Adaboost Boosting: {:.3f}'.format(ada_reg.score(X_test, y_test)))
```

```
KNN regressor score on Train Set after Adaboost Boosting: 0.920
KNN regressor score on Test Set after Adaboost Boosting: 0.645
```

Adaboosting Ridge

Adaboost is used on ridge with c=100 and gamma =0.1 with n_estimators =100 and learning rate =0.1

In [37]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.svm import SVR

ada_reg = AdaBoostRegressor(SVR(C=100, gamma=0.1), n_estimators=100, learning_rate=0.1, random_state=0)
ada_reg.fit(X_train, y_train)
print("Accuracy on training set: {:.4f}".format(ada_reg.score(X_train, y_train)))
print("Accuracy on test set: {:.4f}".format(ada_reg.score(X_test, y_test)))
```

```
Accuracy on training set: 0.9116
Accuracy on test set: 0.8252
```

Task 3: Gradient Boosting

Gradient boosting is a technique which minimizes the overall prediction error. The key idea is to set the target outcomes for the next best model in order to minimize the error. Gradient boosting model is primarily used with decision tree.

The key parameters used are n_estimators and learning_rate

Gradient Boosting Model: Decision Tree

Grid search is used to find best parameters for gradient boosting

In [38]:
```
# Grid search to find the best parameters for gradient boosting
from sklearn.ensemble import GradientBoostingRegressor
parameters = {'n_estimators': [50,100,200,500],
              'learning_rate': [.01, .05, .1, 1]}

grid_search = GridSearchCV(GradientBoostingRegressor(random_state = 0), param_grid = parameters, scoring='neg_mean_squared_error', cv=5,n_jobs= -1)
grid_search = grid_search.fit(X_train,y_train)
print("Grid search best score:", grid_search.best_score_)
print("Grid search best parameters:", grid_search.best_params_)
```

```
Grid search best score: -6.680462083347264
Grid search best parameters: {'learning_rate': 0.05, 'n_estimators': 500}
```

The best parameters n_estimators=500 and learning rate =0.05 with max_depth of 10 is used

In [39]:
```
from  sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=10, n_estimators=500, learning_rate=0.05, random_state=0)
gbrt.fit(X_train, y_train)
print("Accuracy on training set: {:.4f}".format(gbrt.score(X_train, y_train)))
print("Accuracy on test set: {:.4f}".format(gbrt.score(X_test, y_test)))
```

```
Accuracy on training set: 1.0000
Accuracy on test set: 0.9382
```

Task 4: Principal Component Analysis

PCA is statistical technique using dimensionality reduction with sole basis that large number of features add to high dimensionality which essentially leads to overfitting.

PCA thus reduces the number of feautures taken into consideration.

PCA on the dataset for regression

In [40]:
```
X_train
```

Out[40]:
```
array([[1.63434903e-01, 5.88235294e-04, 3.76819709e-01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [2.02216066e-01, 0.00000000e+00, 5.99104143e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.67590028e-01, 5.88235294e-04, 3.35946249e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [2.35457064e-02, 0.00000000e+00, 1.67413214e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.97506925e-01, 4.70588235e-03, 4.16013438e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.71745152e-01, 0.00000000e+00, 1.59574468e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Taken n_components=0.95 are used to explain the variance

In [41]:
```
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV

pca_regression = PCA(n_components = .95)
X_train_reg = pca_regression.fit_transform(X_train)
X_test_reg = pca_regression.transform(X_test)
```

In [42]:
```
pca_regression.n_components_
```

Out[42]:
```
23
```

Explained Variance ratio for each remaining 23 variables

In [43]:
```
pca_regression.explained_variance_ratio_
```

Out[43]:
```
array([0.20454167, 0.07470829, 0.04507021, 0.04365511, 0.04140714,
       0.03964402, 0.03860043, 0.03730578, 0.03685259, 0.03630303,
       0.0357672 , 0.03529331, 0.03482874, 0.0344799 , 0.03426064,
       0.0338621 , 0.03286831, 0.03190215, 0.02177202, 0.01806329,
       0.01654281, 0.01329749, 0.0123916 ])
```

In [44]:
```
1 - pca_regression.explained_variance_ratio_.sum()
```

Out[44]:
```
0.046582160821008745
```

In [45]:
```
X_train_reg.shape
```

Out[45]:
```
(1151, 23)
```

In [46]:
```
X_train_org.shape
```

Out[46]:
```
(1151, 36)
```

In [47]:
```
y_train.shape
```

Out[47]:
```
(1151,)
```

In [48]:
```
X_test_org.shape
```

Out[48]:
```
(384, 36)
```

In [49]:
```
y_test.shape
```

Out[49]:
```
(384,)
```

Model 1 - KNN Regressor

KNN Regressor is run after PCA. Gridsearch is used to identify the best neighbours 1-10

In [50]:
```
from sklearn.neighbors import KNeighborsRegressor

param_grid={'n_neighbors':range(1,10)}

grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=3, return_train_score=True)
grid_search.fit(X_train_reg,y_train)

knn_ts=grid_search.score(X_test_reg, y_test)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
```

```
Best parameters: {'n_neighbors': 6}
Best cross-validation score: 0.6921
```

In [51]:
```
print("Train:",grid_search.best_score_)
print("Test",knn_ts)
print("parameters",grid_search.best_params_)
```

```
Train: 0.6920993007514648
Test 0.7381808648476186
parameters {'n_neighbors': 6}
```

As we can see, the best K value for K nearest neighbors is 6. The KNN is then run with neighbours=6

In [52]:
```
knn = KNeighborsRegressor(n_neighbors=6)
knn.fit(X_train_reg, y_train)
print('Train score on best parameters for KNN Regressor {:.4f}'.format(knn.score(X_train_reg,y_train)))
print('Test score on best parameters for KNN Regressor {:.4f}'.format(knn.score(X_test_reg,y_test)))
```

```
Train score on best parameters for KNN Regressor 0.8011
Test score on best parameters for KNN Regressor 0.7382
```

Now we append the scores to a dataframe called 'post_pca' which stands for 'POST PCA Model_Results'

In [53]:
```
post_pca = pd.DataFrame(columns=('S.No','Model_Name','Parameters', 'Train_Score', 'Test_Score'))
```

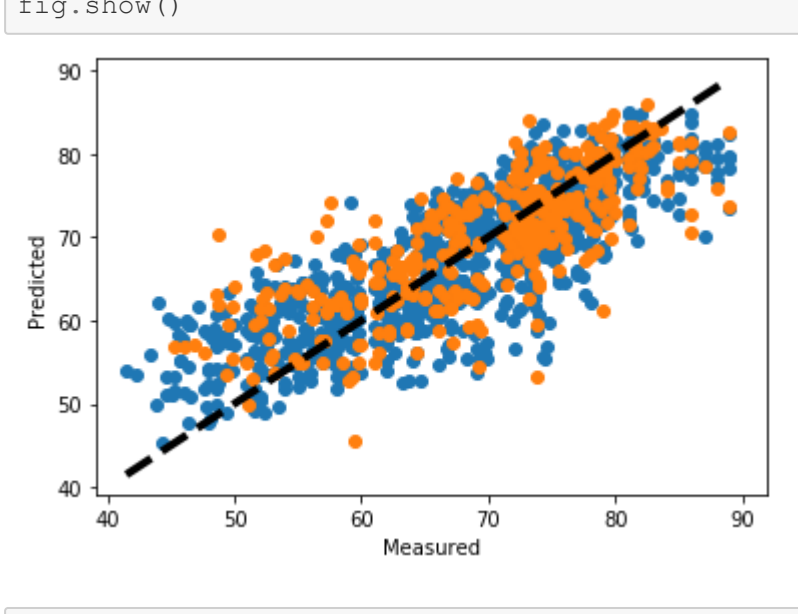
In [54]:
```
post_pca.loc[len(post_pca)]=[1,'KNN Regression','{n_neighbors:6}',knn.score(X_train_reg,y_train),knn.score(X_test_reg,y_test)]
```

In [55]:
```
post_pca
```

Out[55]:

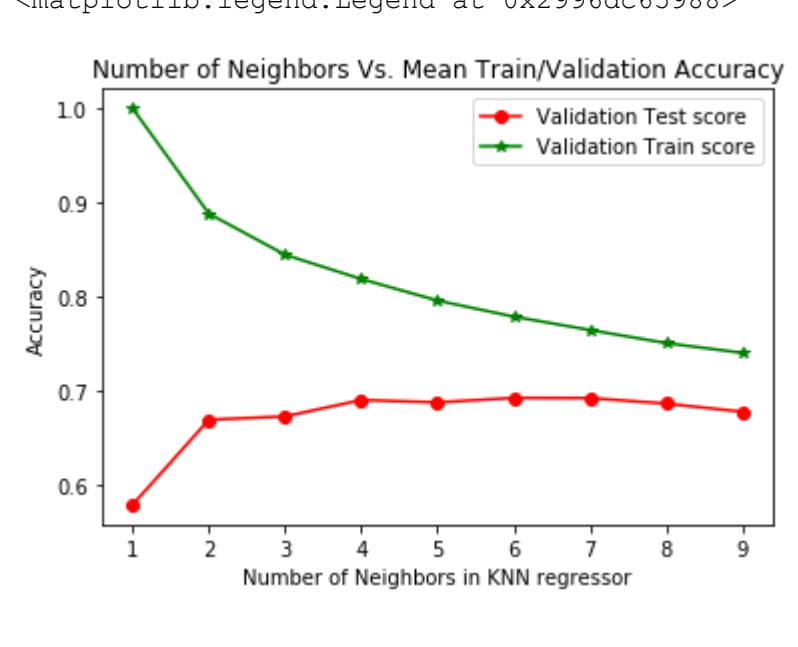
	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.80111	0.738181

In [56]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train_reg,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test_reg,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```



In [57]:
```
results = pd.DataFrame(grid_search.cv_results_)
plt.plot(results['param_n_neighbors'],results['mean_test_score'],marker='o',c='r',label='Validation Test score')
plt.plot(results['param_n_neighbors'],results['mean_train_score'],marker='*',c='g',label='Validation Train score')
plt.title('Number of Neighbors Vs. Mean Train/Validation Accuracy')
plt.xlabel('Number of Neighbors in KNN regressor')
plt.ylabel('Accuracy')
plt.legend()
```

Out[57]:
```
<matplotlib.legend.Legend at 0x2996dc65988>
```



Model 2- Linear Regression

Linear Regression is run on the dataset after PCA. Cross validation score is found for the model

In [58]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

lreg = LinearRegression()
lreg.fit(X_train_reg, y_train)
cv_train=cross_val_score(lreg, X_train_reg, y_train, cv=5)
test=lreg.score(X_test_reg, y_test)
print('Cross val score: %.4f'%cross_val_score(lreg, X_train_reg, y_train, cv=5).mean())
print('Train score: %.4f'%lreg.score(X_train_reg, y_train))
print('Test score: %.4f'%lreg.score(X_test_reg, y_test))
```

```
Cross val score: 0.7678
Train score: 0.7768
Test score: 0.7954
```

In [59]:
```
post_pca.loc[len(post_pca)]=[2,'Linear Regression','none',lreg.score(X_train_reg, y_train),test]
```

In [60]:
```
post_pca
```

Out[60]:

	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.801110	0.738181
1	2	Linear Regression	none	0.776759	0.795437

Model 3 - Ridge Regression

Ridge model is run on the dataset after PCA. Gridsearch is used to identify best alpha

In [61]:
```
from  sklearn.linear_model import Ridge

param_grid={'alpha':[0.001,0.01, 0.1, 1, 3, 5,10, 12,15,20, 100]}

grid_search = GridSearchCV(Ridge(random_state = 0), param_grid, cv=5, return_train_score=True)
grid_search.fit(X_train_reg,y_train)

print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
```

```
Best parameters: {'alpha': 0.1}
Best cross-validation score: 0.7679
```

The best parameters alpha=0.1 is used for the ridge model

In [62]:
```
ridge = Ridge(alpha=0.1)
ridge.fit(X_train_reg, y_train)
print('Train score on best parameters for Ridge regressor {:.4f}'.format(ridge.score(X_train_reg,y_train)))
print('Test score on best parameters for Ridge regressor {:.4f}'.format(ridge.score(X_test_reg,y_test)))
```

```
Train score on best parameters for Ridge regressor 0.7768
Test score on best parameters for Ridge regressor 0.7954
```

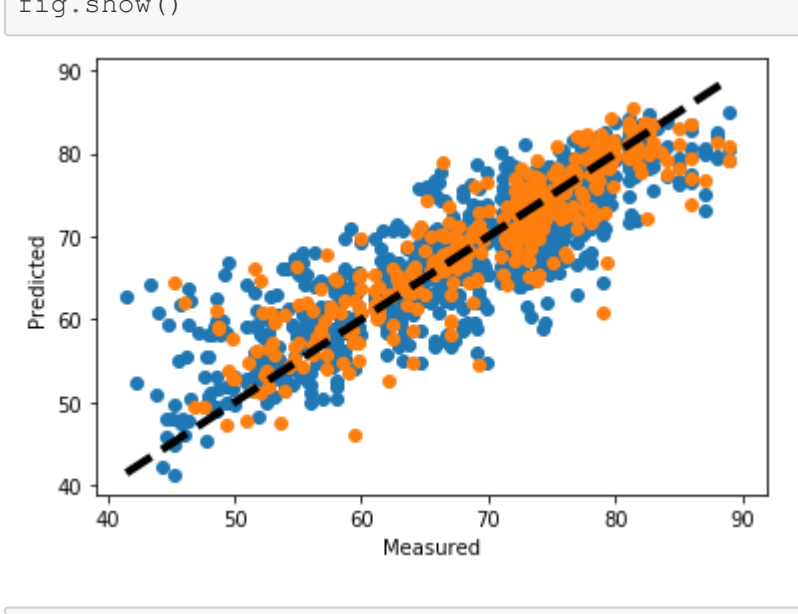
In [63]:
```
post_pca.loc[len(post_pca)]=[3,'Ridge Regression','{alpha:0.1}',ridge.score(X_train_reg,y_train),ridge.score(X_test_reg,y_test)]
```

In [64]:
```
post_pca
```

Out[64]:

	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.801110	0.738181
1	2	Linear Regression	none	0.776759	0.795437
2	3	Ridge Regression	{alpha:0.1}	0.776757	0.795400

In [65]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train_reg,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test_reg,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```



In [66]:
```
results = pd.DataFrame(grid_search.cv_results_)
```

In [67]:
```
results.columns
```

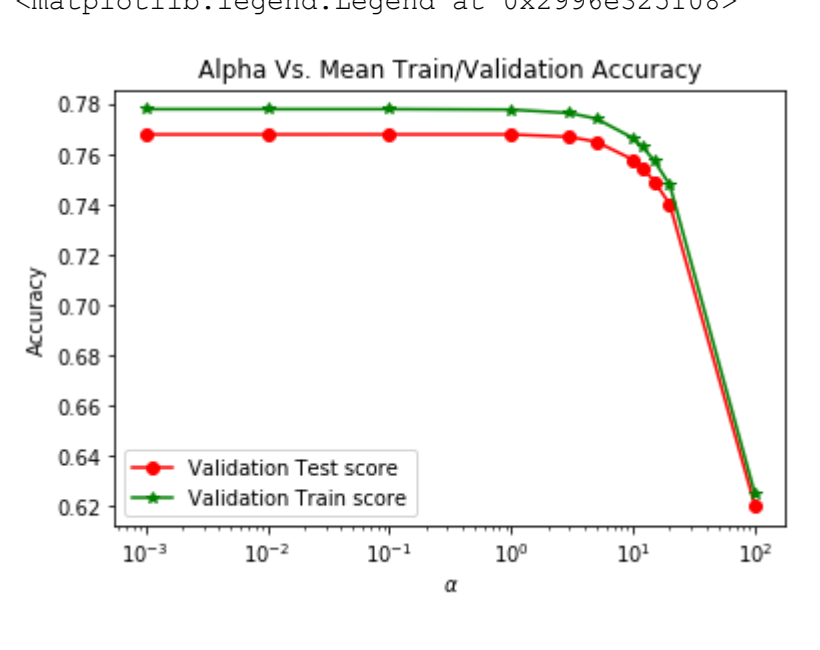
Out[67]:
```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_alpha', 'params', 'split0_test_score', 'split1_test_score',
       'split2_test_score', 'split3_test_score', 'split4_test_score',
       'mean_test_score', 'std_test_score', 'rank_test_score',
       'split0_train_score', 'split1_train_score', 'split2_train_score',
       'split3_train_score', 'split4_train_score', 'mean_train_score',
       'std_train_score'],
      dtype='object')
```

In [68]:
```
grid_search
```

Out[68]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=0,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.001, 0.01, 0.1, 1, 3, 5, 10, 12, 15, 20,
                                   100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [69]:
```
results = pd.DataFrame(grid_search.cv_results_)
plt.plot(results['param_alpha'],results['mean_test_score'],marker='o',c='r',label='Validation Test score')
plt.plot(results['param_alpha'],results['mean_train_score'],marker='*',c='g',label='Validation Train score')
plt.title('Alpha Vs. Mean Train/Validation Accuracy')
plt.xscale('log')
plt.xlabel(r'$\alpha$')
plt.ylabel('Accuracy')
plt.legend()
```

Out[69]:
```
<matplotlib.legend.Legend at 0x2996e325108>
```



Model 4 - Lasso Regression

Lasso model is run on the dataset after PCA. Gridsearch is used to identify best alpha

In [70]:
```
from sklearn.linear_model import Lasso

param_grid={'alpha': [1000,100,10,1,0.1,0.01,0.001]}
grid_search = GridSearchCV(Lasso(random_state = 0), param_grid, cv=5, return_train_score=True)
grid_search.fit(X_train_reg,y_train)
grid_search_test=grid_search.score(X_test_reg,y_test)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
```

```
Best parameters: {'alpha': 0.01}
Best cross-validation score: 0.7683
```

alpha=0.01 is chosen as the best parameter to be used

In [71]:
```
results = pd.DataFrame(grid_search.cv_results_)
lasso = Lasso(alpha=0.01)
lasso.fit(X_train_reg, y_train)
print('Train score on best parameters for Lasso Regressor {:.4f}'.format(lasso.score(X_train_reg,y_train)))
print('Test score on best parameters for lasso Regressor {:.4f}'.format(lasso.score(X_test_reg,y_test)))
```

```
Train score on best parameters for Lasso Regressor 0.7763
Test score on best parameters for lasso Regressor 0.7969
```

In [72]:
```
post_pca.loc[len(post_pca)]=[4,'Lasso Regression','{alpha:0.01}',lasso.score(X_train_reg,y_train),lasso.score(X_test_reg,y_test)]
```

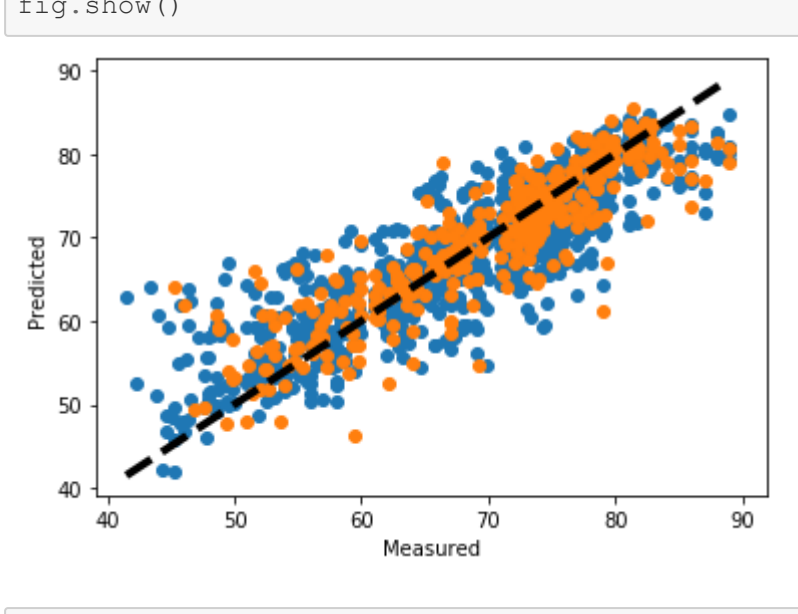
In [73]:
```
post_pca
```

Out[73]:

	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.801110	0.738181
1	2	Linear Regression	none	0.776759	0.795437
2	3	Ridge Regression	{alpha:0.1}	0.776757	0.795400
3	4	Lasso Regression	{alpha:0.01}	0.776288	0.796898

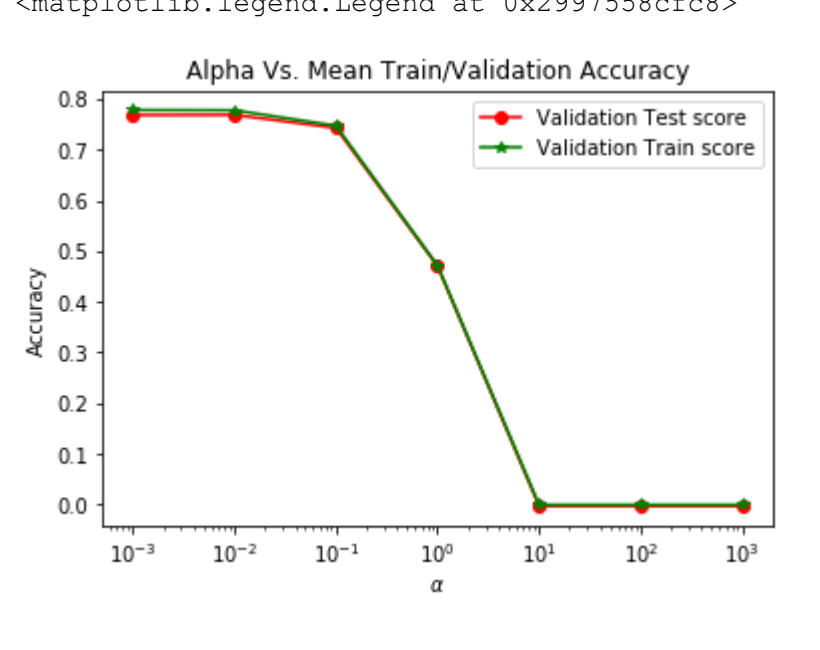
We can now understand by plotting the relation between the predicted values and observed values.

In [74]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train_reg,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test_reg,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```



In [75]:
```
plt.plot(results['param_alpha'],results['mean_test_score'],marker='o',c='r',label='Validation Test score')
plt.plot(results['param_alpha'],results['mean_train_score'],marker='*',c='g',label='Validation Train score')
plt.title('Alpha Vs. Mean Train/Validation Accuracy')
plt.xscale('log')
plt.xlabel(r'$\alpha$')
plt.ylabel('Accuracy')
plt.legend()
```

Out[75]:
```
<matplotlib.legend.Legend at 0x2997558cfc8>
```



Model 5 - Decision Tree

Decision tree model is run on the dataset after PCA. Gridsearch is used to identify best max_depth

In [76]:
```
# decision tree with parameters
from sklearn.tree import DecisionTreeRegressor

dtree_clf = DecisionTreeRegressor(random_state=0)
parameters = { 'max_depth' : [1,2,3,4,5,6,7,8,9,10]}
```

In [77]:
```
#decision tree
grid_search = GridSearchCV(dtree_clf,parameters, cv=10, return_train_score=True)
grid_search.fit(X_train, y_train)
```

Out[77]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse',
                                             max_depth=None, max_features=None,
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             presort='deprecated',
                                             random_state=0, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [78]:
```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_depth': 6}
Best cross-validation score: 0.87
```

Best parameters max_depth = 6 is used with decision tree

In [79]:
```
# decision tree with parameters
dtree = DecisionTreeRegressor(max_depth=6)
dtree.fit(X_train_reg, y_train)

print("Accuracy on training set: {:.4f}".format(dtree.score(X_train_reg, y_train)))
print("Accuracy on test set: {:.4f}".format(dtree.score(X_test_reg, y_test)))
```

```
Accuracy on training set: 0.8483
Accuracy on test set: 0.6891
```

In [80]:
```
train_s_list = []
test_s_list = []
train_s_list.append(dtree.score(X_train_reg, y_train))
test_s_list.append(dtree.score(X_test_reg, y_test))
post_pca.loc[len(post_pca)]=[5,'Decision Tree','{maxdepth:6}',train_s_list[0],test_s_list[0]]
```

Model 6 - Polynomial Regression

The best value of features is identified between 1,2 for polynomial regression.

In [81]:
```
from  sklearn.preprocessing  import PolynomialFeatures
train_s_list = []
test_s_list = []
for n in range(1,6):

        poly = PolynomialFeatures(n)
        X_train_reg_poly = poly.fit_transform(X_train_reg)
        X_test_reg_poly = poly.transform(X_test_reg)
        lreg.fit(X_train_reg_poly, y_train)
        train_s_list.append(lreg.score(X_train_reg_poly, y_train))
        test_s_list.append(lreg.score(X_test_reg_poly, y_test))
```

In [82]:
```
print(train_s_list)
print(test_s_list)
```

```
[0.776758744147749, 0.7310441931588437, 1.0, 1.0, 1.0]
[0.7954367495086329, 0.33910878658241284, -265.006414345935, -1.5610986056354168, -1.486224164617230
5]
```

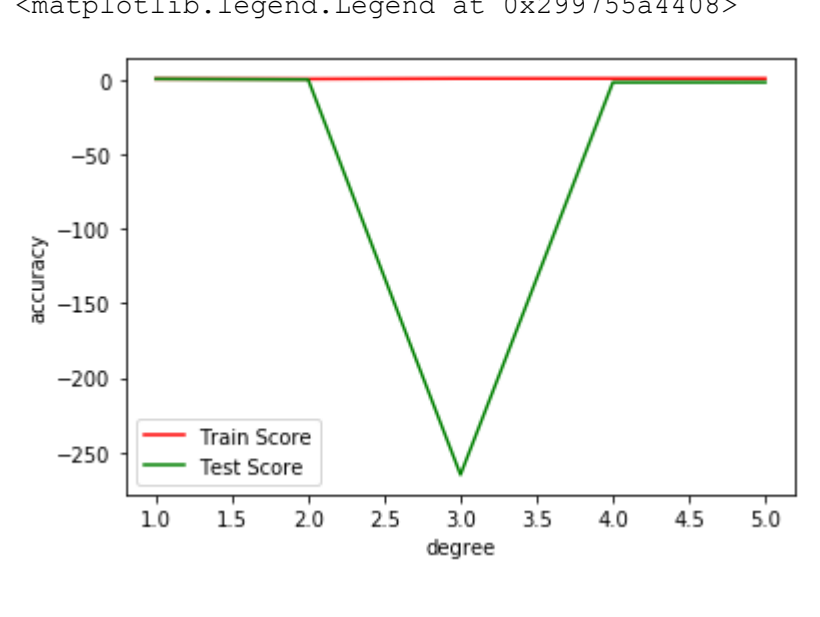
The best scores are rendered by polynomials of degree 1.

In [83]:
```
print("Train",train_s_list[0])
print("Test",test_s_list[0])
print("Best Parameters n=",1)
```

```
Train 0.776758744147749
Test 0.7954367495086329
Best Parameters n= 1
```

In [84]:
```
x_axis = range(1,6)
plt.plot(x_axis, train_s_list, c = 'r', label = 'Train Score')
plt.plot(x_axis, test_s_list, c = 'g', label = 'Test Score')

plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()
```

Out[84]:
```
<matplotlib.legend.Legend at 0x299755a4408>
```



Adding the scores to the model results

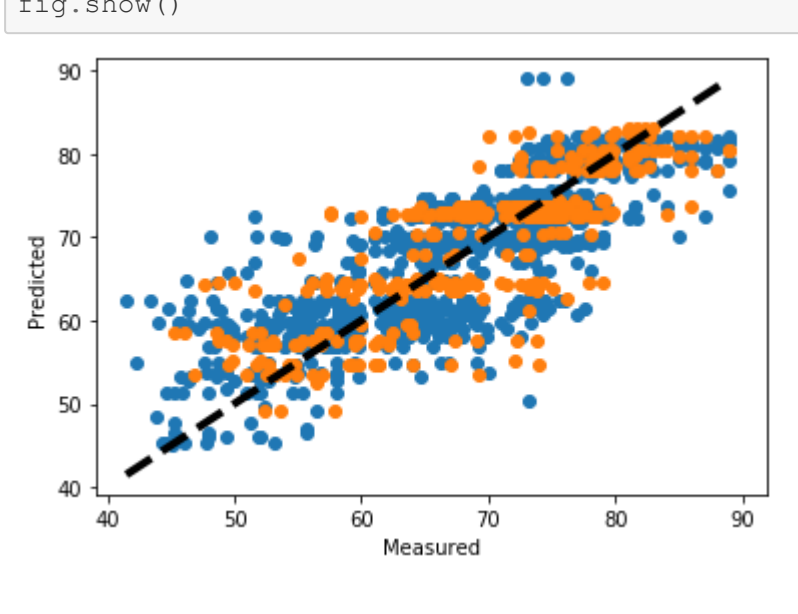
In [85]:
```
post_pca.loc[len(post_pca)]=[6,'Polynomial Regression','{degree:1}',train_s_list[0],test_s_list[0]]
```

In [86]:
```
post_pca
```

Out[86]:

	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.801110	0.738181
1	2	Linear Regression	none	0.776759	0.795437
2	3	Ridge Regression	{alpha:0.1}	0.776757	0.795400
3	4	Lasso Regression	{alpha:0.01}	0.776288	0.796898
4	5	Decision Tree	{maxdepth:6}	0.848306	0.689063
5	6	Polynomial Regression	{degree:1}	0.776759	0.795437

In [87]:
```
from sklearn.model_selection import cross_val_predict
fig,ax = plt.subplots()
predicted = cross_val_predict(grid_search,X_train_reg,y_train, cv=5)
ax.scatter(y_train, predicted)
predicted = cross_val_predict(grid_search,X_test_reg,y_test, cv=5)
ax.scatter(y_test, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
fig.show()
```



Model 7 - Support Vector Machine (LinearSVR, SVR Kernel - Linear, Rbf and Poly)

The support vector machine model is run on the dataset after pca with linear svr , svr with kernel set as linear, rbf and poly

In [88]:
```
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [0.1, 1, 10, 100], 'degree':[1,3,5]}

grid_search = GridSearchCV(SVR(), param_grid, cv=5, return_train_score=True)
grid_search.fit(X_train_reg, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
```

```
Best parameters: {'C': 100, 'degree': 1, 'gamma': 0.1}
Best cross-validation score: 0.8030
```

The best parameters are c:100 degree:1 gamma:0.1

In [89]:
```
clf1 = LinearSVR(C=100).fit(X_train_reg, y_train)
clf2 = SVR(kernel='linear', C=100).fit(X_train_reg, y_train)
clf3 = SVR(kernel='rbf', gamma=0.1, C=100).fit(X_train_reg, y_train)
clf4 = SVR(kernel='poly', degree=1, C=100).fit(X_train_reg, y_train)
print('Train score on best parameters for LinearSVR - {:.4f}'.format(clf1.score(X_train_reg,y_train)))
print('Test score on best parameters for LinearSVR - {:.4f}'.format(clf1.score(X_test_reg,y_test)))
print('Train score on best parameters for SVR kernel - Linear {:.4f}'.format(clf2.score(X_train_reg,y_train)))
print('Test score on best parameters for SVR kernel - Linear {:.4f}'.format(clf2.score(X_test_reg,y_test)))
print('Train score on best parameters for SVR kernel - rbf {:.4f}'.format(clf3.score(X_train_reg,y_train)))
print('Test score on best parameters for SVR kernel - rbf {:.4f}'.format(clf3.score(X_test_reg,y_test)))
print('Train score on best parameters for SVR kernel - poly {:.4f}'.format(clf4.score(X_train_reg,y_train)))
print('Test score on best parameters for SVR kernel - poly {:.4f}'.format(clf4.score(X_test_reg,y_test)))
```

```
Train score on best parameters for LinearSVR - 0.7706
Test score on best parameters for LinearSVR - 0.7933
Train score on best parameters for SVR kernel - Linear 0.7706
Test score on best parameters for SVR kernel - Linear 0.7935
Train score on best parameters for SVR kernel - rbf 0.8418
Test score on best parameters for SVR kernel - rbf 0.8258
Train score on best parameters for SVR kernel - poly 0.7705
Test score on best parameters for SVR kernel - poly 0.7934
```

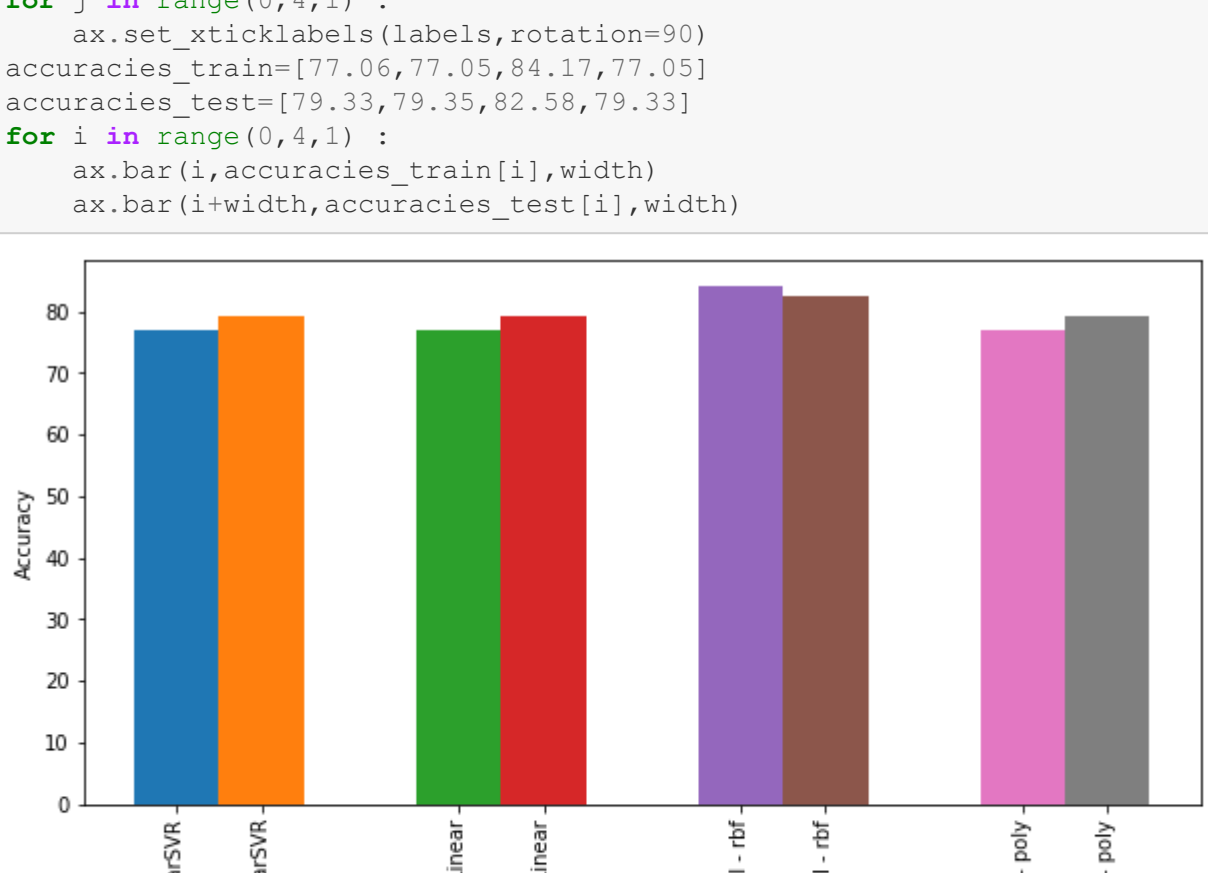
In [90]:
```
post_pca.loc[len(post_pca)]=[7,'Linear SVC','{C: 100}',clf1.score(X_train_reg, y_train),clf1.score(X_test_reg, y_test)]
post_pca.loc[len(post_pca)]=[8,'SVC Kernel- Linear','{C: 100}',clf2.score(X_train_reg, y_train),clf2.score(X_test_reg, y_test)]
post_pca.loc[len(post_pca)]=[9,'SVC Kernel - RBF','{C: 100, gamma: 0.1}',clf3.score(X_train_reg, y_train),clf3.score(X_test_reg, y_test)]
post_pca.loc[len(post_pca)]=[10,'SVC Kernal -Polynomial','{C: 100, degree: 1}',clf4.score(X_train_reg, y_train),clf4.score(X_test_reg, y_test)]
```

In [91]:
```
post_pca
```

Out[91]:

	S.No	Model_Name	Parameters	Train_Score	Test_Score
0	1	KNN Regression	{n_neighbors:6}	0.801110	0.738181
1	2	Linear Regression	none	0.776759	0.795437
2	3	Ridge Regression	{alpha:0.1}	0.776757	0.795400
3	4	Lasso Regression	{alpha:0.01}	0.776288	0.796898
4	5	Decision Tree	{maxdepth:6}	0.848306	0.689063
5	6	Polynomial Regression	{degree:1}	0.776759	0.795437
6	7	Linear SVC	{C: 100}	0.770616	0.793329
7	8	SVC Kernel- Linear	{C: 100}	0.770555	0.793507
8	9	SVC Kernel - RBF	{C: 100, gamma: 0.1}	0.841785	0.825815
9	10	SVC Kernal -Polynomial	{C: 100, degree: 1}	0.770524	0.793376

```
In [92]: fig, ax = plt.subplots(figsize=(10,5))
width = 0.3
plt.xlabel('SVR')
plt.ylabel('Accuracy')
labels = ['Train LinearSVR','Test LinearSVR','Train SVR kernel - Linear','Test SVR kernel - Linear','Train SVR kernel - rbf',
          'Test SVR kernel - rbf','Train SVR kernel - poly','Test SVR kernel - poly']
list_ticks = [0,.3,1,1.3,2,2.3,3,3.3]
ax.set_xticks(list_ticks)
for j in range(0,4,1) :
    ax.set_xticklabels(labels,rotation=90)
accuracies_train=[77.06,77.05,84.17,77.05]
accuracies_test=[79.33,79.35,82.58,79.33]
for i in range(0,4,1) :
    ax.bar(i,accuracies_train[i],width)
    ax.bar(i+width,accuracies_test[i],width)
```



The train and test scores of various SVR are displayed for the kernels are displayed, we find that rbf kernel is not overfitting.

Comparing accuracies for Regression models before and after PCA

The accuracies of the models before and after PCA are compared.

```
In [93]: # Index Levels
outside = ['Before PCA','Before PCA','After PCA','After PCA']
inside = ['Training Accuracy','Test Accuracy','Training Accuracy','Test Accuracy']
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

```
In [94]: data_reg = np.array([(0.8407,0.8141,0.8131,0.8049,0.9925,0.9825,0.8052,0.8056,0.8874,0.9358),
                     (0.7362,0.8050,0.8051,0.8060,0.9001,0.8100,0.8073,0.8053,0.8437,0.8130),
                     (0.8011,0.7767,0.7767,0.7762,0.8483,0.7767,0.7703,0.7705 ,0.8417, 0.7705),
 (0.7381,0.7954,0.7954,0.7968,0.6893,0.7954,0.7926,0.7935, 0.8258, 0.7933)])
```

```
In [95]: df = pd.DataFrame(data_reg,index=hier_index,columns=['KNN','Linear','Ridge','Lasso','Decision Tree','Po
lynomial','LinearSVR','SVR Kernel - Linear','SVR Kernel - RBF','SVR Kernel - Poly'])
```

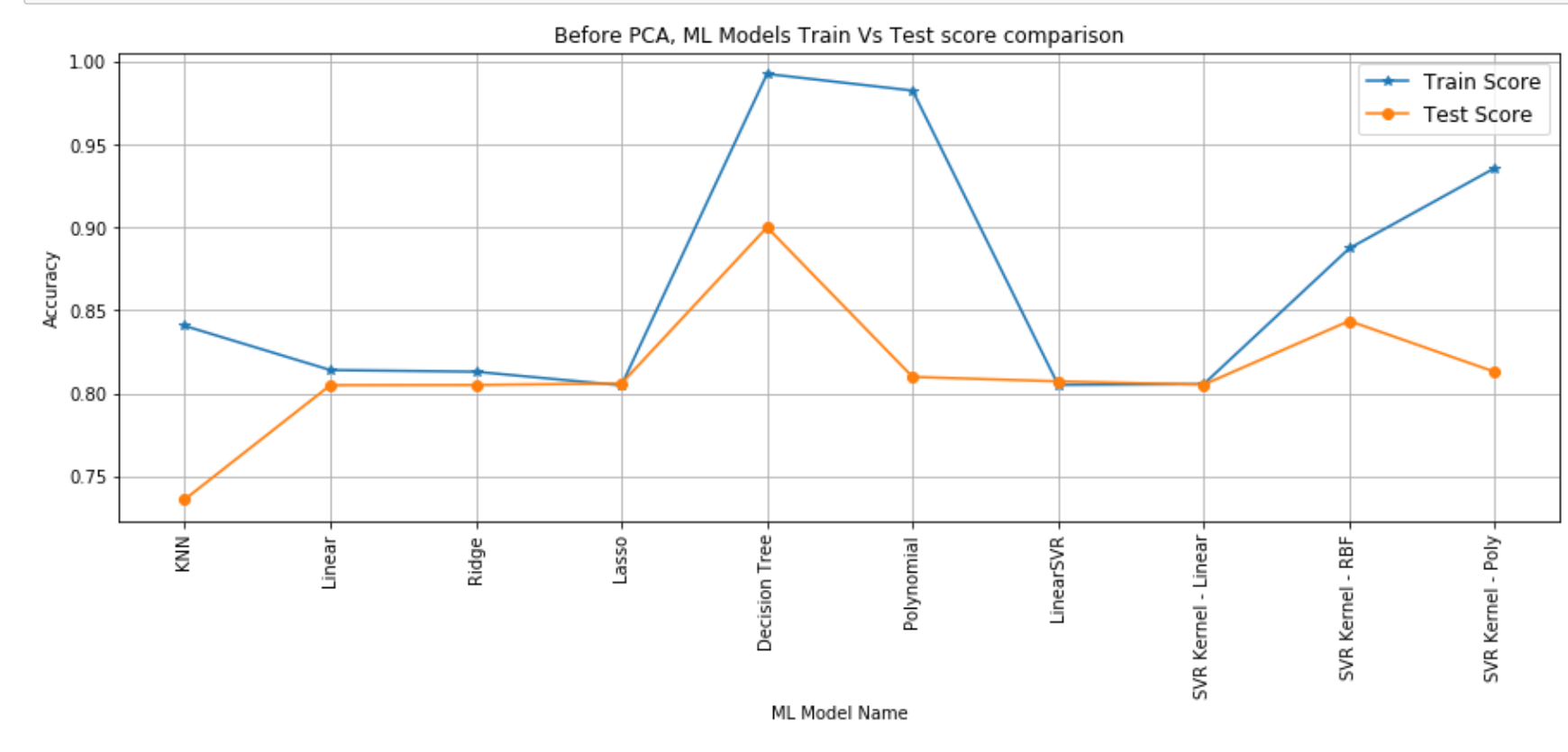
```
In [96]: Models=['KNN','Linear','Ridge','Lasso','Decision Tree','Polynomial','LinearSVR','SVR Kernel - Linear',
'SVR Kernel - RBF','SVR Kernel - Poly']
```

```
In [97]: #import seaborn as sns

#cm = sns.light_palette("#2ecc71", as_cmap=True)

#s = df.style.background_gradient()
df
```

Out[97]:

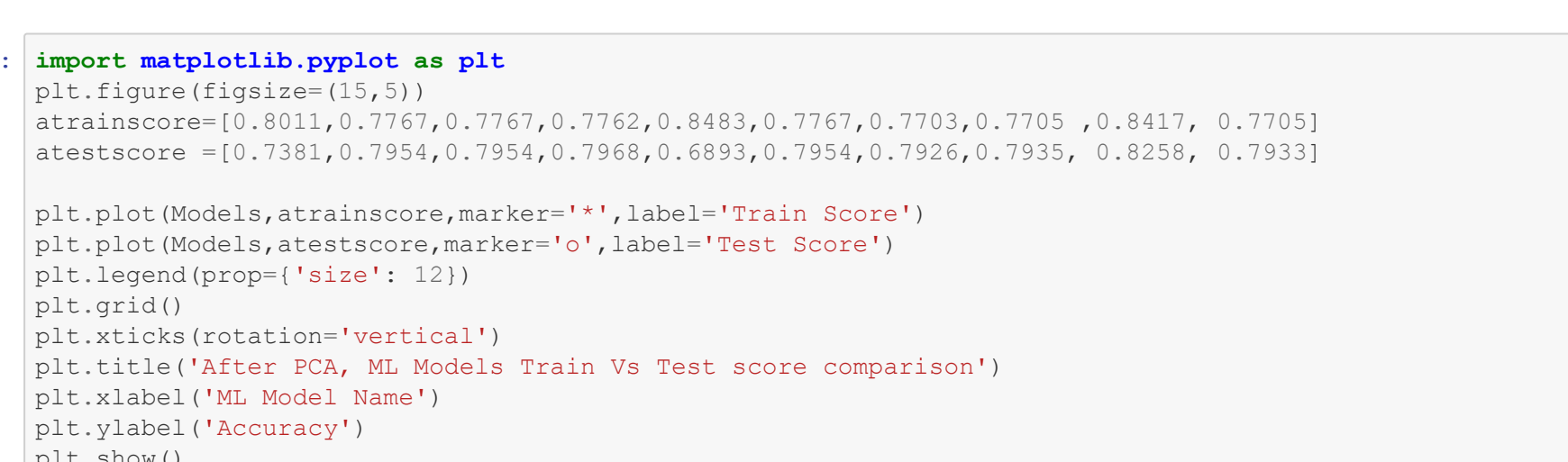
		KNN	Linear	Ridge	Lasso	Decision Tree	Polynomial	LinearSVR	SVR Kernel - Linear	SVR Kernel - RBF	SVR Kernel - Poly
Before PCA	Training Accuracy	0.8407	0.8141	0.8131	0.8049	0.9925	0.9825	0.8052	0.8056	0.8874	0.9358
	Test Accuracy	0.7362	0.8050	0.8051	0.8060	0.9001	0.8100	0.8073	0.8053	0.8437	0.8130
After PCA	Training Accuracy	0.8011	0.7767	0.7767	0.7762	0.8483	0.7767	0.7703	0.7705	0.8417	0.7705
	Test Accuracy	0.7381	0.7954	0.7954	0.7968	0.6893	0.7954	0.7926	0.7935	0.8258	0.7933

```
In [98]: import matplotlib.pyplot as plt
btrainscore=[0.8407,0.8141,0.8131,0.8049,0.9925,0.9825,0.8052,0.8056,0.8874,0.9358]
btestscore =[0.7362,0.8050,0.8051,0.8060,0.9001,0.8100,0.8073,0.8053,0.8437,0.8130]
plt.figure(figsize=(15,5))
plt.plot(Models,btrainscore,marker='*',label='Train Score')
plt.plot(Models,btestscore,marker='o',label='Test Score')
plt.legend(prop={'size': 12})
plt.grid()
plt.xticks(rotation='vertical')
plt.title('Before PCA, ML Models Train Vs Test score comparison')
plt.xlabel('ML Model Name')
plt.ylabel('Accuracy')
plt.show()
```



Before PCA, Decision tree had a good performance with less train test difference

```
In [99]: import matplotlib.pyplot as plt
plt.figure(figsize=(15,5))
atrainscore=[0.8011,0.7767,0.7767,0.7762,0.8483,0.7767,0.7703,0.7705 ,0.8417, 0.7705]
atestscore =[0.7381,0.7954,0.7954,0.7968,0.6893,0.7954,0.7926,0.7935, 0.8258, 0.7933]

plt.plot(Models,atrainscore,marker='*',label='Train Score')
plt.plot(Models,atestscore,marker='o',label='Test Score')
plt.legend(prop={'size': 12})
plt.grid()
plt.xticks(rotation='vertical')
plt.title('After PCA, ML Models Train Vs Test score comparison')
plt.xlabel('ML Model Name')
plt.ylabel('Accuracy')
plt.show()
```



After PCA, SVR Kernel RBF has less train test score difference

Inference Obtained:

1. For Regression, after using PCA we see that the difference between the test and train has reduced with SVR kernel poly having a better t srain test difference
2. Comparing the best model before PCA (Decision tree) to the best model post PCA (SVM RBF), we can see that the Post-PCA model had a marginally lower test score
3. Here, we were able to reduce the number of features which also explains the loss in predictive power. Whether we should use PCA or not depends on the cost of computation saved and the business cost incurred in marginally losing predictive power
4. As dimensions present are low , PCA didnt have much boosting present.

The scores achieved without PCA is better for most models but the train test score difference has reduced after PCA.

Without PCA the best model is Decision Tree with train - 0.9925 test - 0.9001

With PCA the best model is SVM RBF with train - 0.8417 test - 0.8258

Task 5 : Deep-Learning Models

```
In [100]: from keras.models import Sequential
from keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(10)
```

```
In [101]: #step 1: build model
model1 = Sequential()
#input layer
model1.add(Dense(30, input_dim = 36, activation = 'relu'))
#hidden layers
model1.add(Dense(20, activation = 'relu'))
model1.add(Dense(10, activation = 'relu'))
#output layer
model1.add(Dense(1))

#step 2: make computational graph - compile
model1.compile(loss= 'mse' , optimizer = 'adam',metrics = ['mse'] )

#step 3: train the model - fit
model1.fit(X_train, y_train, epochs = 100, batch_size = 20);
```

```
Epoch 1/100
58/58 [==============================] - 0s 1ms/step - loss: 4812.8203 - mse: 4812.8203
Epoch 2/100
58/58 [==============================] - 0s 1ms/step - loss: 4136.8467 - mse: 4136.8467
Epoch 3/100
58/58 [==============================] - 0s 1ms/step - loss: 1293.7776 - mse: 1293.7776
Epoch 4/100
58/58 [==============================] - 0s 1ms/step - loss: 74.2012 - mse: 74.2012
Epoch 5/100
58/58 [==============================] - 0s 1ms/step - loss: 63.0099 - mse: 63.0099
Epoch 6/100
58/58 [==============================] - 0s 1ms/step - loss: 57.5845 - mse: 57.5845
Epoch 7/100
58/58 [==============================] - 0s 1ms/step - loss: 53.0962 - mse: 53.0962
Epoch 8/100
58/58 [==============================] - 0s 1ms/step - loss: 49.6791 - mse: 49.6791
Epoch 9/100
58/58 [==============================] - 0s 1ms/step - loss: 46.6352 - mse: 46.6352
Epoch 10/100
58/58 [==============================] - 0s 1ms/step - loss: 43.3724 - mse: 43.3724
Epoch 11/100
58/58 [==============================] - 0s 1ms/step - loss: 40.5106 - mse: 40.5106
Epoch 12/100
58/58 [==============================] - 0s 1ms/step - loss: 37.9665 - mse: 37.9665
Epoch 13/100
58/58 [==============================] - 0s 1ms/step - loss: 35.2124 - mse: 35.2124
Epoch 14/100
58/58 [==============================] - 0s 1ms/step - loss: 33.3009 - mse: 33.3009
Epoch 15/100
58/58 [==============================] - 0s 1ms/step - loss: 30.6373 - mse: 30.6373
Epoch 16/100
58/58 [==============================] - 0s 1ms/step - loss: 28.7451 - mse: 28.7451
Epoch 17/100
58/58 [==============================] - 0s 1ms/step - loss: 26.7478 - mse: 26.7478
Epoch 18/100
58/58 [==============================] - 0s 1ms/step - loss: 25.3024 - mse: 25.3024
Epoch 19/100
58/58 [==============================] - 0s 1ms/step - loss: 23.8417 - mse: 23.8417
Epoch 20/100
58/58 [==============================] - 0s 1ms/step - loss: 22.5908 - mse: 22.5908
Epoch 21/100
58/58 [==============================] - 0s 1ms/step - loss: 21.7379 - mse: 21.7379
Epoch 22/100
58/58 [==============================] - 0s 1ms/step - loss: 21.1908 - mse: 21.1908
Epoch 23/100
58/58 [==============================] - 0s 1ms/step - loss: 20.1850 - mse: 20.1850
Epoch 24/100
58/58 [==============================] - 0s 1ms/step - loss: 19.6855 - mse: 19.6855
Epoch 25/100
58/58 [==============================] - 0s 1ms/step - loss: 19.1937 - mse: 19.1937
Epoch 26/100
58/58 [==============================] - 0s 1ms/step - loss: 18.6977 - mse: 18.6977
Epoch 27/100
58/58 [==============================] - 0s 1ms/step - loss: 18.3258 - mse: 18.3258
Epoch 28/100
58/58 [==============================] - 0s 1ms/step - loss: 18.2658 - mse: 18.2658
Epoch 29/100
58/58 [==============================] - 0s 1ms/step - loss: 17.7482 - mse: 17.7482
Epoch 30/100
58/58 [==============================] - 0s 1ms/step - loss: 17.5110 - mse: 17.5110
Epoch 31/100
58/58 [==============================] - 0s 1ms/step - loss: 17.2439 - mse: 17.2439
Epoch 32/100
58/58 [==============================] - 0s 1ms/step - loss: 17.0508 - mse: 17.0508
Epoch 33/100
58/58 [==============================] - 0s 1ms/step - loss: 16.8623 - mse: 16.8623
Epoch 34/100
58/58 [==============================] - 0s 1ms/step - loss: 16.7400 - mse: 16.7400
Epoch 35/100
58/58 [==============================] - 0s 1ms/step - loss: 16.5130 - mse: 16.5130
Epoch 36/100
58/58 [==============================] - 0s 1ms/step - loss: 16.5904 - mse: 16.5904
Epoch 37/100
58/58 [==============================] - 0s 1ms/step - loss: 16.3956 - mse: 16.3956
Epoch 38/100
58/58 [==============================] - 0s 1ms/step - loss: 16.4771 - mse: 16.4771
Epoch 39/100
58/58 [==============================] - 0s 1ms/step - loss: 16.1339 - mse: 16.1339
Epoch 40/100
58/58 [==============================] - 0s 1ms/step - loss: 15.9465 - mse: 15.9465
Epoch 41/100
58/58 [==============================] - 0s 1ms/step - loss: 15.7530 - mse: 15.7530
Epoch 42/100
58/58 [==============================] - 0s 1ms/step - loss: 15.8240 - mse: 15.8240
Epoch 43/100
58/58 [==============================] - 0s 1ms/step - loss: 15.5673 - mse: 15.5673
Epoch 44/100
58/58 [==============================] - 0s 1ms/step - loss: 15.4267 - mse: 15.4267
Epoch 45/100
58/58 [==============================] - 0s 1ms/step - loss: 15.4141 - mse: 15.4141
Epoch 46/100
58/58 [==============================] - 0s 1ms/step - loss: 15.3985 - mse: 15.3985
Epoch 47/100
58/58 [==============================] - 0s 1ms/step - loss: 15.2180 - mse: 15.2180
Epoch 48/100
58/58 [==============================] - 0s 1ms/step - loss: 15.1490 - mse: 15.1490
Epoch 49/100
58/58 [==============================] - 0s 1ms/step - loss: 15.1851 - mse: 15.1851
Epoch 50/100
58/58 [==============================] - ETA: 0s - loss: 15.3510 - mse: 15.351 - 0s 1ms/step - loss:
14.9255 - mse: 14.9255
Epoch 51/100
58/58 [==============================] - 0s 1ms/step - loss: 14.8785 - mse: 14.8785
Epoch 52/100
58/58 [==============================] - 0s 1ms/step - loss: 14.8394 - mse: 14.8394
Epoch 53/100
58/58 [==============================] - 0s 1ms/step - loss: 14.6701 - mse: 14.6701
Epoch 54/100
58/58 [==============================] - 0s 1ms/step - loss: 14.4956 - mse: 14.4956
Epoch 55/100
58/58 [==============================] - 0s 1ms/step - loss: 15.1085 - mse: 15.1085
Epoch 56/100
58/58 [==============================] - 0s 1ms/step - loss: 14.9785 - mse: 14.9785
Epoch 57/100
58/58 [==============================] - 0s 1ms/step - loss: 14.7095 - mse: 14.7095
Epoch 58/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2839 - mse: 14.2839
Epoch 59/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2416 - mse: 14.2416
Epoch 60/100
58/58 [==============================] - 0s 1ms/step - loss: 14.3793 - mse: 14.3793
Epoch 61/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2440 - mse: 14.2440
Epoch 62/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2545 - mse: 14.2545
Epoch 63/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2644 - mse: 14.2644
Epoch 64/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2819 - mse: 14.2819
Epoch 65/100
58/58 [==============================] - 0s 1ms/step - loss: 14.3360 - mse: 14.3360
Epoch 66/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0664 - mse: 14.0664
Epoch 67/100
58/58 [==============================] - 0s 1ms/step - loss: 14.1134 - mse: 14.1134
Epoch 68/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0154 - mse: 14.0154
Epoch 69/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9722 - mse: 13.9722
Epoch 70/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9941 - mse: 13.9941
Epoch 71/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6379 - mse: 13.6379
Epoch 72/100
58/58 [==============================] - 0s 1ms/step - loss: 13.7123 - mse: 13.7123
Epoch 73/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9282 - mse: 13.9282
Epoch 74/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6956 - mse: 13.6956
Epoch 75/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0402 - mse: 14.0402
Epoch 76/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5379 - mse: 13.5379
Epoch 77/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5294 - mse: 13.5294
Epoch 78/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5034 - mse: 13.5034
Epoch 79/100
58/58 [==============================] - 0s 1ms/step - loss: 13.4671 - mse: 13.4671
Epoch 80/100
58/58 [==============================] - 0s 1ms/step - loss: 13.4961 - mse: 13.4961
Epoch 81/100
58/58 [==============================] - 0s 1ms/step - loss: 13.2912 - mse: 13.2912
Epoch 82/100
58/58 [==============================] - 0s 1ms/step - loss: 13.3040 - mse: 13.3040
Epoch 83/100
58/58 [==============================] - 0s 1ms/step - loss: 13.3111 - mse: 13.3111
Epoch 84/100
58/58 [==============================] - 0s 1ms/step - loss: 13.2350 - mse: 13.2350
Epoch 85/100
58/58 [==============================] - 0s 1ms/step - loss: 13.4190 - mse: 13.4190
Epoch 86/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0780 - mse: 13.0780
Epoch 87/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6607 - mse: 13.6607
Epoch 88/100
58/58 [==============================] - 0s 1ms/step - loss: 14.4405 - mse: 14.4405
Epoch 89/100
58/58 [==============================] - 0s 1ms/step - loss: 13.1822 - mse: 13.1822
Epoch 90/100
58/58 [==============================] - 0s 1ms/step - loss: 13.1242 - mse: 13.1242
Epoch 91/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0909 - mse: 13.0909
Epoch 92/100
58/58 [==============================] - 0s 1ms/step - loss: 13.1985 - mse: 13.1985
Epoch 93/100
58/58 [==============================] - 0s 1ms/step - loss: 13.1826 - mse: 13.1826
Epoch 94/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0136 - mse: 13.0136
Epoch 95/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8753 - mse: 12.8753
Epoch 96/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6514 - mse: 13.6514
Epoch 97/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8837 - mse: 12.8837
Epoch 98/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8375 - mse: 12.8375
Epoch 99/100
58/58 [==============================] - 0s 1ms/step - loss: 12.7875 - mse: 12.7875
Epoch 100/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8098 - mse: 12.8098
```

Perceptron model evaulation metrics:

the perceptron model is evaluated using the train and test values

```
In [102]: model1.evaluate(X_train, y_train)
36/36 [==============================] - 0s 1ms/step - loss: 12.7929 - mse: 12.7929
Out[102]: [12.792928695678711, 12.792928695678711]
```

```
In [103]: model1.evaluate(X_test, y_test)
12/12 [==============================] - 0s 2ms/step - loss: 15.3448 - mse: 15.3448
Out[103]: [15.344801902770996, 15.344801902770996]
```

R Squared score for perceptron

R2 score for both train and test are calculated. It determines the variance in the dependent variable

```
In [104]: from sklearn.metrics import r2_score

y_train_predict = model1.predict(X_train)
y_test_predict = model1.predict(X_test)

print('Train score: {:.4f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.4f}'.format(r2_score(y_test, y_test_predict)))

Train score: 0.8569
Test score: 0.8206
```

Single layer Perceptron with Gridsearch

```
In [105]: def create_model():
    #step 1: build model
    model = Sequential()
    #input layer
    model.add(Dense(20, input_dim = 36, activation = 'relu'))
    #output layer
    model.add(Dense(1))

    #step 2: make computational graph - compile
    model.compile(loss= 'mse' ,optimizer = 'adam', metrics = ['mse'] )
    return model
```

```
In [106]: from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasRegressor

model = KerasRegressor(build_fn = create_model, verbose = 0)

param_grid = {'batch_size':[20,40,100] , 'epochs':[50, 100]}
grid_search = GridSearchCV(estimator= model, param_grid = param_grid, cv = 5)
```

```
In [107]: import logging
import tensorflow as tf
tf.get_logger().setLevel(logging.ERROR)
grid_search_result = grid_search.fit(X_train, y_train)
```

```
In [108]: print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

Best parameters: {'batch_size': 20, 'epochs': 100}
Best cross-validation score: -30.33
```

```
In [109]: # Step 3: Fit the model
model.fit(X_train, y_train, epochs=100, batch_size=20)
Out[109]: <tensorflow.python.keras.callbacks.History at 0x2990c7ac4c8>
```

R Squared score for perceptron

```
In [110]: from sklearn.metrics import r2_score

y_train_predict = model.predict(X_train)
y_test_predict = model.predict(X_test)

print('Train score: {:.4f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.4f}'.format(r2_score(y_test, y_test_predict)))

Train score: 0.7920
Test score: 0.7842
```

2. Multi layer perceptron

```
In [111]: import tensorflow as tf
from tensorflow import keras as ker
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

```
In [112]: #step 1: build model
model1 = Sequential()
#input layer
model1.add(Dense(30, input_dim = 36, activation = 'relu'))
#hidden layers
model1.add(Dense(20, activation = 'relu'))
model1.add(Dense(10, activation = 'relu'))
#output layer
model1.add(Dense(1))

#step 2: make computational graph - compile
model1.compile(loss= 'mse' , optimizer = 'adam',metrics = ['mse'] )

#step 3: train the model - fit
model1.fit(X_train, y_train, epochs = 100, batch_size = 20);
```

```
Epoch 1/100
58/58 [==============================] - 0s 1ms/step - loss: 4737.7314 - mse: 4737.7314
Epoch 2/100
58/58 [==============================] - 0s 1ms/step - loss: 3878.9666 - mse: 3878.9666
Epoch 3/100
58/58 [==============================] - 0s 1ms/step - loss: 966.5458 - mse: 966.5458
Epoch 4/100
58/58 [==============================] - 0s 1ms/step - loss: 71.6891 - mse: 71.6891
Epoch 5/100
58/58 [==============================] - 0s 1ms/step - loss: 62.6237 - mse: 62.6237
Epoch 6/100
58/58 [==============================] - 0s 1ms/step - loss: 57.4776 - mse: 57.4776
Epoch 7/100
58/58 [==============================] - 0s 1ms/step - loss: 52.7391 - mse: 52.7391
Epoch 8/100
58/58 [==============================] - 0s 1ms/step - loss: 48.8241 - mse: 48.8241
Epoch 9/100
58/58 [==============================] - 0s 1ms/step - loss: 45.0931 - mse: 45.0931
Epoch 10/100
58/58 [==============================] - 0s 1ms/step - loss: 41.5219 - mse: 41.5219
Epoch 11/100
58/58 [==============================] - 0s 1ms/step - loss: 38.5708 - mse: 38.5708
Epoch 12/100
58/58 [==============================] - 0s 1ms/step - loss: 35.2724 - mse: 35.2724
Epoch 13/100
58/58 [==============================] - 0s 1ms/step - loss: 32.8241 - mse: 32.8241
Epoch 14/100
58/58 [==============================] - 0s 1ms/step - loss: 30.5929 - mse: 30.5929
Epoch 15/100
58/58 [==============================] - 0s 1ms/step - loss: 28.4570 - mse: 28.4570
Epoch 16/100
58/58 [==============================] - 0s 1ms/step - loss: 26.5964 - mse: 26.5964
Epoch 17/100
58/58 [==============================] - 0s 1ms/step - loss: 25.1068 - mse: 25.1068
Epoch 18/100
58/58 [==============================] - 0s 1ms/step - loss: 23.7383 - mse: 23.7383
Epoch 19/100
58/58 [==============================] - 0s 1ms/step - loss: 22.6578 - mse: 22.6578
Epoch 20/100
58/58 [==============================] - 0s 1ms/step - loss: 21.5147 - mse: 21.5147
Epoch 21/100
58/58 [==============================] - 0s 1ms/step - loss: 20.5158 - mse: 20.5158
Epoch 22/100
58/58 [==============================] - 0s 1ms/step - loss: 19.6454 - mse: 19.6454
Epoch 23/100
58/58 [==============================] - 0s 1ms/step - loss: 18.9663 - mse: 18.9663
Epoch 24/100
58/58 [==============================] - 0s 1ms/step - loss: 18.4016 - mse: 18.4016
Epoch 25/100
58/58 [==============================] - 0s 1ms/step - loss: 17.8359 - mse: 17.8359
Epoch 26/100
58/58 [==============================] - 0s 1ms/step - loss: 17.4990 - mse: 17.4990
Epoch 27/100
58/58 [==============================] - 0s 1ms/step - loss: 16.8953 - mse: 16.8953
Epoch 28/100
58/58 [==============================] - 0s 1ms/step - loss: 16.6055 - mse: 16.6055
Epoch 29/100
58/58 [==============================] - 0s 1ms/step - loss: 16.5566 - mse: 16.5566
Epoch 30/100
58/58 [==============================] - 0s 1ms/step - loss: 16.2024 - mse: 16.2024
Epoch 31/100
58/58 [==============================] - 0s 1ms/step - loss: 15.8008 - mse: 15.8008
Epoch 32/100
58/58 [==============================] - 0s 1ms/step - loss: 15.7782 - mse: 15.7782
Epoch 33/100
58/58 [==============================] - 0s 1ms/step - loss: 15.7129 - mse: 15.7129
Epoch 34/100
58/58 [==============================] - 0s 1ms/step - loss: 15.4591 - mse: 15.4591
Epoch 35/100
58/58 [==============================] - 0s 1ms/step - loss: 15.3591 - mse: 15.3591
Epoch 36/100
58/58 [==============================] - 0s 1ms/step - loss: 15.1801 - mse: 15.1801
Epoch 37/100
58/58 [==============================] - 0s 1ms/step - loss: 15.0250 - mse: 15.0250
Epoch 38/100
58/58 [==============================] - 0s 1ms/step - loss: 15.2275 - mse: 15.2275
Epoch 39/100
58/58 [==============================] - 0s 1ms/step - loss: 14.9080 - mse: 14.9080
Epoch 40/100
58/58 [==============================] - 0s 1ms/step - loss: 14.7864 - mse: 14.7864
Epoch 41/100
58/58 [==============================] - 0s 1ms/step - loss: 14.7447 - mse: 14.7447
Epoch 42/100
58/58 [==============================] - 0s 1ms/step - loss: 14.9269 - mse: 14.9269
Epoch 43/100
58/58 [==============================] - 0s 1ms/step - loss: 14.8113 - mse: 14.8113
Epoch 44/100
58/58 [==============================] - 0s 1ms/step - loss: 14.5818 - mse: 14.5818
Epoch 45/100
58/58 [==============================] - 0s 1ms/step - loss: 14.6751 - mse: 14.6751
Epoch 46/100
58/58 [==============================] - 0s 1ms/step - loss: 14.3587 - mse: 14.3587
Epoch 47/100
58/58 [==============================] - 0s 1ms/step - loss: 14.2916 - mse: 14.2916
Epoch 48/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0903 - mse: 14.0903
Epoch 49/100
58/58 [==============================] - 0s 1ms/step - loss: 14.6735 - mse: 14.6735
Epoch 50/100
58/58 [==============================] - 0s 1ms/step - loss: 14.3190 - mse: 14.3190
Epoch 51/100
58/58 [==============================] - 0s 1ms/step - loss: 14.1699 - mse: 14.1699
Epoch 52/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9571 - mse: 13.9571
Epoch 53/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9797 - mse: 13.9797
Epoch 54/100
58/58 [==============================] - 0s 1ms/step - loss: 13.9458 - mse: 13.9458
Epoch 55/100
58/58 [==============================] - 0s 1ms/step - loss: 13.8641 - mse: 13.8641
Epoch 56/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0188 - mse: 14.0188
Epoch 57/100
58/58 [==============================] - 0s 1ms/step - loss: 13.7736 - mse: 13.7736
Epoch 58/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6779 - mse: 13.6779
Epoch 59/100
58/58 [==============================] - 0s 1ms/step - loss: 13.7927 - mse: 13.7927
Epoch 60/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5646 - mse: 13.5646
Epoch 61/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5372 - mse: 13.5372
Epoch 62/100
58/58 [==============================] - 0s 1ms/step - loss: 14.0073 - mse: 14.0073
Epoch 63/100
58/58 [==============================] - 0s 1ms/step - loss: 13.4162 - mse: 13.4162
Epoch 64/100
58/58 [==============================] - 0s 1ms/step - loss: 13.4573 - mse: 13.4573
Epoch 65/100
58/58 [==============================] - 0s 1ms/step - loss: 13.3572 - mse: 13.3572
Epoch 66/100
58/58 [==============================] - 0s 1ms/step - loss: 13.2981 - mse: 13.2981
Epoch 67/100
58/58 [==============================] - 0s 1ms/step - loss: 13.3400 - mse: 13.3400
Epoch 68/100
58/58 [==============================] - 0s 1ms/step - loss: 13.3688 - mse: 13.3688
Epoch 69/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5324 - mse: 13.5324
Epoch 70/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6775 - mse: 13.6775
Epoch 71/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5251 - mse: 13.5251
Epoch 72/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0999 - mse: 13.0999
Epoch 73/100
58/58 [==============================] - 0s 1ms/step - loss: 13.6233 - mse: 13.6233
Epoch 74/100
58/58 [==============================] - 0s 1ms/step - loss: 13.2130 - mse: 13.2130
Epoch 75/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0382 - mse: 13.0382
Epoch 76/100
58/58 [==============================] - 0s 1ms/step - loss: 12.9729 - mse: 12.9729
Epoch 77/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8937 - mse: 12.8937
Epoch 78/100
58/58 [==============================] - 0s 1ms/step - loss: 13.5577 - mse: 13.5577
Epoch 79/100
58/58 [==============================] - 0s 1ms/step - loss: 12.9669 - mse: 12.9669
Epoch 80/100
58/58 [==============================] - 0s 1ms/step - loss: 12.7484 - mse: 12.7484
Epoch 81/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0681 - mse: 13.0681
Epoch 82/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8386 - mse: 12.8386
Epoch 83/100
58/58 [==============================] - 0s 1ms/step - loss: 12.7742 - mse: 12.7742
Epoch 84/100
58/58 [==============================] - 0s 1ms/step - loss: 12.9366 - mse: 12.9366
Epoch 85/100
58/58 [==============================] - 0s 1ms/step - loss: 12.6778 - mse: 12.6778
Epoch 86/100
58/58 [==============================] - 0s 1ms/step - loss: 13.0405 - mse: 13.0405
Epoch 87/100
58/58 [==============================] - 0s 1ms/step - loss: 12.8881 - mse: 12.8881
Epoch 88/100
58/58 [==============================] - 0s 1ms/step - loss: 12.5989 - mse: 12.5989
Epoch 89/100
58/58 [==============================] - 0s 1ms/step - loss: 12.6559 - mse: 12.6559
Epoch 90/100
58/58 [==============================] - 0s 1ms/step - loss: 12.3782 - mse: 12.3782
Epoch 91/100
58/58 [==============================] - 0s 1ms/step - loss: 12.6630 - mse: 12.6630
Epoch 92/100
58/58 [==============================] - 0s 993us/step - loss: 12.4548 - mse: 12.4548
Epoch 93/100
58/58 [==============================] - 0s 999us/step - loss: 12.3575 - mse: 12.3575
Epoch 94/100
58/58 [==============================] - 0s 1ms/step - loss: 12.4735 - mse: 12.4735
Epoch 95/100
58/58 [==============================] - 0s 1ms/step - loss: 12.2388 - mse: 12.2388
Epoch 96/100
58/58 [==============================] - 0s 1ms/step - loss: 12.3701 - mse: 12.3701
Epoch 97/100
58/58 [==============================] - 0s 1ms/step - loss: 12.4955 - mse: 12.4955
Epoch 98/100
58/58 [==============================] - 0s 1ms/step - loss: 12.5448 - mse: 12.5448
Epoch 99/100
58/58 [==============================] - 0s 1ms/step - loss: 12.1287 - mse: 12.1287
Epoch 100/100
58/58 [==============================] - 0s 1ms/step - loss: 12.5313 - mse: 12.5313
```

Perceptron model evaulation metrics:

```
In [113]: import logging
tf.get_logger().setLevel(logging.ERROR)
model1.evaluate(X_train, y_train)
36/36 [==============================] - 0s 1ms/step - loss: 11.7170 - mse: 11.7170
Out[113]: [11.716975212097168, 11.716975212097168]
```

```
In [114]: model1.evaluate(X_test, y_test)
12/12 [==============================] - 0s 1ms/step - loss: 13.8205 - mse: 13.8205
Out[114]: [13.820454597473145, 13.820454597473145]
```

R Squared score for perceptron

```
In [115]: from sklearn.metrics import r2_score

y_train_predict = model1.predict(X_train)
y_test_predict = model1.predict(X_test)

print('Train score: {:.4f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.4f}'.format(r2_score(y_test, y_test_predict)))

Train score: 0.8689
Test score: 0.8384
```

Multilayer Perceptron with Gridsearch

```
In [116]: def create_model():
    #step 1: build model
    model = Sequential()
    #input layer
    model.add(Dense(20, input_dim = 36, activation = 'relu'))
    #hidden layers
    model.add(Dense(20, activation = 'relu'))
    model.add(Dense(10, activation = 'relu'))
    #output layer
    model.add(Dense(1))

    #step 2: make computational graph - compile
    model.compile(loss= 'mse' ,optimizer = 'adam', metrics = ['mse'] )
    return model
```

```
In [117]: from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasRegressor

model = KerasRegressor(build_fn = create_model, verbose = 0)

param_grid = {'batch_size':[20,40,100] , 'epochs':[50, 100]}
grid_search = GridSearchCV(estimator= model, param_grid = param_grid, cv = 5)
```

```
In [118]: import logging
import tensorflow as tf
tf.get_logger().setLevel(logging.ERROR)

grid_search_result = grid_search.fit(X_train, y_train)
```

```
In [119]: print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

Best parameters: {'batch_size': 20, 'epochs': 100}
Best cross-validation score: -17.27
```

```
In [120]: # Step 3: Fit the model
model.fit(X_train, y_train, epochs=50, batch_size=20)
Out[120]: <tensorflow.python.keras.callbacks.History at 0x29925592d48>
```

R Squared score for perceptron

```
In [121]: from sklearn.metrics import r2_score

y_train_predict = model.predict(X_train)
y_test_predict = model.predict(X_test)

print('Train score: {:.4f}'.format(r2_score(y_train, y_train_predict)))
print('Test score: {:.4f}'.format(r2_score(y_test, y_test_predict)))

Train score: 0.8169
Test score: 0.8048
```