Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024
ACTIVITY	**Assignment 7.1**
Name	Cuevas, Christian Jay L.
Section	CPE32S3
Date Performed:	4/8/2024
Date Submitted:	4/10/2024
Instructor:	Engr. Roman M. Richard

Instructions

1. Choose any dataset applicable to the classification problem, and also, choose any dataset applicable to the regression problem.
2. Explain your datasets and the problem being addressed.
3. For classification, do the following:
   - Create a base model
   - Evaluate the model with k-fold cross validation
   - Improve the accuracy of your model by applying additional hidden layers
4. For regression, do the following:
   - Create a base model
   - Improve the model by standardizing the dataset
   - Show tuning of layers and neurons (see evaluating small and larger networks)
5. Submit the link to your Google Colab (make sure that it is accessible to me)

Dataset

- There are 2 dataset that was used in this assignment and they are separated by the task that will be performed to them.
- The first dataset is for classification task and it is about the features of a breast mass computed from a digitized image. There are 31 predictors and 1 target variable. The goal of this dataset is to create a model to predict if a person is diagnosed with breast cancer according to the digitized image of their breast mass.
- The second dataset is for regression task and it is about features that contribute to the MPG or Miles per Gallon consumption of a car. This dataset contains 9 features that consists of 1 ID, 7 predictor and 1 target variable. The goal of this dataset is to create a model to predict the MPG of a car based on different factors such as displacement, horsepower, weight, cylinders, acceleration, model year and origin.

Importing Libraries and Dataset

In [1]:

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

In [2]:

```
!pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (
from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packag
es (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (fr
om scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (fro
m scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (fr
om scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packa
ges (from scikit-learn>=1.0.0->scikeras) (3.4.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

In [3]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from keras.optimizers import Adam, SGD, RMSprop
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Activation, Dropout
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_c
urve, accuracy_score
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from scikeras.wrappers import KerasRegressor
from sklearn.pipeline import Pipeline
```

In [4]:

```
from ucimlrepo import fetch_ucirepo

"""Classification Dataset"""
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)

bCancer_df = breast_cancer_wisconsin_diagnostic.data.original
```

In [5]:

```
"""Regression Dataset"""
car_mpg = fetch_ucirepo(id=9)

mpg_df = car_mpg.data.original
```

CLASSIFICATION TASK:

Performing Exploratory Data Analysis on Classification Dataset

- **Let's perform Exploratory Data Analysis for the Breast Cancer Wisconsin dataset that will be used for the classification task.**
- **I will be performing separate Exploratory Data Analysis for the Classification Dataset and the Regression Dataset.**
- **We will first look at the data types of the features and the target variable by using the .info() function.**

In [ ]:

```
bCancer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   569 non-null    int64  
 1   radius1              569 non-null    float64
 2   texture1             569 non-null    float64
 3   perimeter1           569 non-null    float64
 4   area1                569 non-null    float64
 5   smoothness1          569 non-null    float64
 6   compactness1         569 non-null    float64
 7   concavity1           569 non-null    float64
 8   concave_points1      569 non-null    float64
 9   symmetry1            569 non-null    float64
 10  fractal_dimension1   569 non-null    float64
 11  radius2              569 non-null    float64
 12  texture2             569 non-null    float64
 13  perimeter2           569 non-null    float64
 14  area2                569 non-null    float64
 15  smoothness2          569 non-null    float64
 16  compactness2         569 non-null    float64
 17  concavity2           569 non-null    float64
 18  concave_points2      569 non-null    float64
 19  symmetry2            569 non-null    float64
 20  fractal_dimension2   569 non-null    float64
 21  radius3              569 non-null    float64
 22  texture3             569 non-null    float64
 23  perimeter3           569 non-null    float64
 24  area3                569 non-null    float64
 25  smoothness3          569 non-null    float64
 26  compactness3         569 non-null    float64
 27  concavity3           569 non-null    float64
 28  concave_points3      569 non-null    float64
 29  symmetry3            569 non-null    float64
 30  fractal_dimension3   569 non-null    float64
 31  Diagnosis            569 non-null    object 
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Observation:

- **Here we can observe that this dataset has 32 features, 31 predictor and 1 target variable. The entries for this dataset is 569 and I can see that there are no missing values. This dataset consists of 30 float64 data types and 1 int64 data type. There is one object data type which is the "Diagnosis" and it is the target variable for this dataset.**

In [ ]:

```
bCancer_df.isnull().sum()
```

Out[ ]:

```
ID                    0
radius1               0
texture1              0
```

```
perimeter1            0
area1                 0
smoothness1           0
compactness1          0
concavity1            0
concave_points1       0
symmetry1             0
fractal_dimension1    0
radius2               0
texture2              0
perimeter2            0
area2                 0
smoothness2           0
compactness2          0
concavity2            0
concave_points2       0
symmetry2             0
fractal_dimension2    0
radius3               0
texture3              0
perimeter3            0
area3                 0
smoothness3           0
compactness3          0
concavity3            0
concave_points3       0
symmetry3             0
fractal_dimension3    0
Diagnosis             0
dtype: int64
```

Observation:

- **Here we can further check if there are Null values. Since the result for every column is equal to 0, this means that there are no null values.**

In [ ]:

```
bCancer_df.head()
```

Out[ ]:

	ID	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points1	symmetry1	...	te
0	842302	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	
1	842517	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	
2	84300903	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	
3	84348301	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	
4	84358402	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	

5 rows × 32 columns

Observation:

- **To further observe the dataset, we can look at the first 5 rows of this dataset and the corresponding values for each column. From here, we can see that the ranges for the values of the dataset varies, there are some values that ranges from 0.1 - 1 and there are some values that ranges from 77 - 135. We can further observe this by using the .describe() method.**

In [ ]:

```
bCancer_df.describe()
```

Out[ ]:

	ID	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.04891
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.03880
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.00000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.02031
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.03350
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.07400
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.20120

8 rows × 31 columns

Observation:

- **Here we can see that the .describe() method returned count, mean, std, min, 25%, 50%, 75%, and max. In the row of "count", we can see that all columns have 569 entries. We can observe that the mean of each column varies and the gap between them maybe small or large. The std measures the dispersion of the values, we can see that most of the std are small values which describes low standard deviation that means they are near the mean or the data is clustered. Now we can also observe the min to max values of this dataset. We can see that the minimium and maximum values for each column represents the range of the values. The 25%, 50% and 75% shows us the distribution of the data between the range of min and max.**

- **To proceed with checking the correlation and plotting the heatmap, we need to first remove the column "ID" since this is just a unique identifier that will not contribute to the prediction of the class.**

In [ ]:

```
bCancer_df = bCancer_df.drop(["ID"], axis = 1)
```

In [ ]:

```
bCancer_df.head()
```

Out[ ]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points1	symmetry1	fractal_dimensio
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.078
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.056
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.059
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.097
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.058

5 rows × 31 columns

- **I will also change the class "M" and "B", to 0 and 1 respectively using lambda function. "M" stands for**

Malignant and "B" stands for Benign.

In [ ]:

```
bCancer_df["Diagnosis"] = bCancer_df["Diagnosis"].apply(lambda toLabel: 0 if toLabel ==
  'M' else 1)
```

In [ ]:

```
bCancer_df.head(5)
```

Out[ ]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points1	symmetry1	fractal_dimensio
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.078
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.056
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.059
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.097
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.058

5 rows × 31 columns

In [ ]:

```
bCancer_df.corr(method="pearson", numeric_only = True)
```

Out[ ]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points1	symme
radius1	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147
texture1	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071
perimeter1	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183
area1	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151
smoothness1	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557
compactness1	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602
concavity1	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500
concave_points1	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462
symmetry1	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000
fractal_dimension1	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479
radius2	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050	0.303
texture2	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480	0.128
perimeter2	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650	0.313
area2	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299	0.223
smoothness2	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653	0.187
compactness2	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424	0.421
concavity2	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167	0.342
concave_points2	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634	0.393
symmetry2	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351	0.449
fractal_dimension2	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584	0.331

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_points1	symme
radius3	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.830318	0.185
texture3	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.292752	0.090
perimeter3	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.855923	0.219
area3	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.809630	0.177
smoothness3	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.452753	0.426
compactness3	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.667454	0.473
concavity3	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.752399	0.433
concave_points3	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.910155	0.430
symmetry3	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.375744	0.699
fractal_dimension3	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.368661	0.438
Diagnosis	-0.730029	-0.415185	-0.742636	-0.708984	-0.358560	-0.596534	-0.696360	-0.776614	-0.330

31 rows × 31 columns

In [ ]:

```
bCancer_df.corr(method="pearson", numeric_only = True)['Diagnosis'].sort_values(ascending=
False)
```

Out[ ]:

```
Diagnosis             1.000000
smoothness2           0.067016
fractal_dimension1    0.012838
texture2              0.008303
symmetry2             0.006522
fractal_dimension2   -0.077972
concavity2           -0.253730
compactness2         -0.292999
fractal_dimension3   -0.323872
symmetry1            -0.330499
smoothness1          -0.358560
concave_points2      -0.408042
texture1             -0.415185
symmetry3            -0.416294
smoothness3          -0.421465
texture3             -0.456903
area2                -0.548236
perimeter2           -0.556141
radius2              -0.567134
compactness3         -0.590998
compactness1         -0.596534
concavity3           -0.659610
concavity1           -0.696360
area1                -0.708984
radius1              -0.730029
area3                -0.733825
perimeter1           -0.742636
radius3              -0.776454
concave_points1      -0.776614
perimeter3           -0.782914
concave_points3      -0.793566
Name: Diagnosis, dtype: float64
```

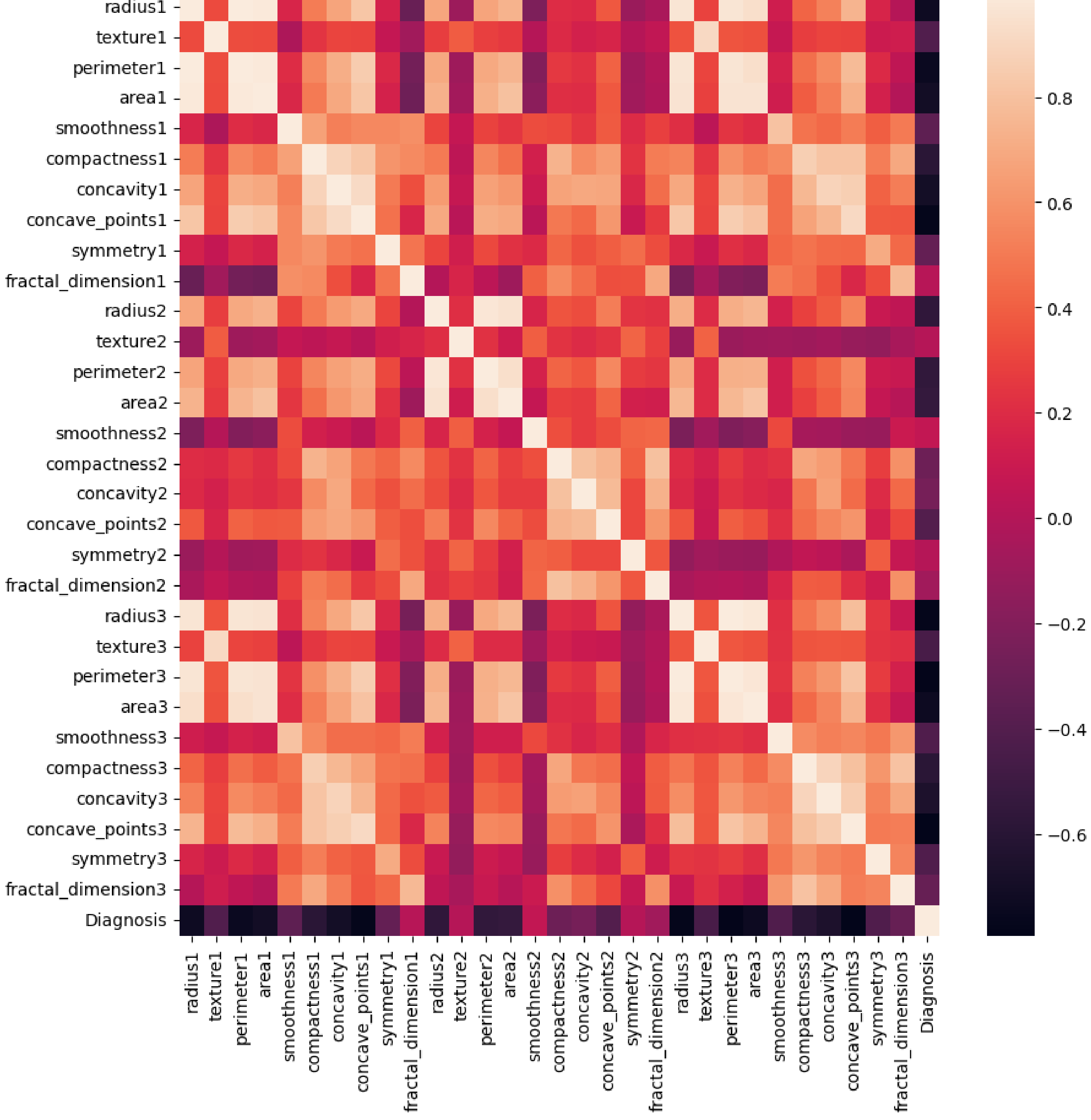
In [ ]:

```
fig = plt.figure(figsize=(10,10), dpi=100)

sns.heatmap(bCancer_df.corr())
```

Out[ ]:

```
<Axes: >
```



Observation:

- **We can observe here that the values of the correlation of the features to the target variable is low and close to 0. This means that there are no variables with a strong relationship to our predictor variable and it may affect the performance of our model. The feature with the highest coefficient with our target variable is the "smoothness2", but it only has the value of 0.06.**

Preparing the data and Splitting the dataset into Training, Validation and Testing sets (Classification)

- **To prevent any bias in our data, I will be performing normalization of the data using StandardScaler(). I will also be splitting the data into 60/20/20 [1,2].**

```
In [ ]:
```

```
X = bCancer_df.drop(["Diagnosis"], axis = 1)
y = bCancer_df["Diagnosis"]

X, X_test, y, y_test = train_test_split(X, y, test_size =0.2, random_state=100)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size =0.2, random_state=110)
```

In [ ]:

```
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_val_norm = scaler.transform(X_val)
X_test_norm = scaler.transform(X_test)
```

In [ ]:

```
X_train_norm.shape, X_test_norm.shape, X_val_norm.shape
```

Out[ ]:

```
((364, 30), (114, 30), (91, 30))
```

In [ ]:

```
y_train.shape, y_test.shape, y_val.shape
```

Out[ ]:

```
((364,), (114,), (91,))
```

Training the Base Model for Classification

- **Let's build the model using keras. I will use 1 hidden layer with 20 hidden nodes. The activation function that I will use for the hidden layer is "relu" as it is the most recommended [3] and "sigmoid" for the output layer. I will use Adam optimizer with the learning rate of 0.001, and binary crossentropy for the loss function.**
- **I used 20 hidden nodes because I followed the rule of thumb that the hidden layer nodes should be 2/3 of the input layer nodes [4].**
- **I used 30 epochs arbitrarily for the starting number of epoch, this can increase or decrease later depending on the result or the graph that we will be observing.**

In [ ]:

```
def cl_baseline_model():

  model = tf.keras.models.Sequential([
      tf.keras.layers.Dense(20, input_shape=(30,), activation = "relu"),
      tf.keras.layers.Dense(1, activation = "sigmoid")
  ])

  model.compile(Adam(learning_rate = 0.001),
                loss = "binary_crossentropy",
                metrics=["accuracy"]
                )
  return model

clmodel = cl_baseline_model()

bCancer_model1 = clmodel.fit(X_train_norm, y_train, validation_data = (X_val_norm, y_val)
,epochs = 30)
```

```
Epoch 1/30
12/12 [==============================] - 1s 21ms/step - loss: 0.7915 - accuracy: 0.4918 - v
al_loss: 0.6001 - val_accuracy: 0.6593
Epoch 2/30
12/12 [==============================] - 0s 6ms/step - loss: 0.6113 - accuracy: 0.6786 - va
l_loss: 0.4731 - val_accuracy: 0.7582
Epoch 3/30
12/12 [==============================] - 0s 5ms/step - loss: 0.4839 - accuracy: 0.8049 - va
l_loss: 0.3921 - val_accuracy: 0.8352
Epoch 4/30
12/12 [==============================] - 0s 6ms/step - loss: 0.3969 - accuracy: 0.8681 - va
```

```
12/12 [==============================] - 0s 6ms/step - loss: 0.3969 - accuracy: 0.8681 - va
l_loss: 0.3370 - val_accuracy: 0.8901
Epoch 5/30
12/12 [==============================] - 0s 6ms/step - loss: 0.3318 - accuracy: 0.8984 - va
l_loss: 0.2973 - val_accuracy: 0.9011
Epoch 6/30
12/12 [==============================] - 0s 6ms/step - loss: 0.2842 - accuracy: 0.9231 - va
l_loss: 0.2667 - val_accuracy: 0.9011
Epoch 7/30
12/12 [==============================] - 0s 6ms/step - loss: 0.2484 - accuracy: 0.9341 - va
l_loss: 0.2420 - val_accuracy: 0.9011
Epoch 8/30
12/12 [==============================] - 0s 5ms/step - loss: 0.2210 - accuracy: 0.9423 - va
l_loss: 0.2249 - val_accuracy: 0.9121
Epoch 9/30
12/12 [==============================] - 0s 5ms/step - loss: 0.1991 - accuracy: 0.9478 - va
l_loss: 0.2117 - val_accuracy: 0.9231
Epoch 10/30
12/12 [==============================] - 0s 6ms/step - loss: 0.1821 - accuracy: 0.9505 - va
l_loss: 0.2020 - val_accuracy: 0.9231
Epoch 11/30
12/12 [==============================] - 0s 5ms/step - loss: 0.1673 - accuracy: 0.9533 - va
l_loss: 0.1925 - val_accuracy: 0.9231
Epoch 12/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1553 - accuracy: 0.9588 - va
l_loss: 0.1846 - val_accuracy: 0.9451
Epoch 13/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1446 - accuracy: 0.9560 - va
l_loss: 0.1763 - val_accuracy: 0.9451
Epoch 14/30
12/12 [==============================] - 0s 9ms/step - loss: 0.1355 - accuracy: 0.9615 - va
l_loss: 0.1712 - val_accuracy: 0.9451
Epoch 15/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1274 - accuracy: 0.9670 - va
l_loss: 0.1665 - val_accuracy: 0.9451
Epoch 16/30
12/12 [==============================] - 0s 8ms/step - loss: 0.1206 - accuracy: 0.9698 - va
l_loss: 0.1627 - val_accuracy: 0.9451
Epoch 17/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1144 - accuracy: 0.9725 - va
l_loss: 0.1593 - val_accuracy: 0.9451
Epoch 18/30
12/12 [==============================] - 0s 6ms/step - loss: 0.1087 - accuracy: 0.9780 - va
l_loss: 0.1556 - val_accuracy: 0.9451
Epoch 19/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1038 - accuracy: 0.9780 - va
l_loss: 0.1530 - val_accuracy: 0.9451
Epoch 20/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0993 - accuracy: 0.9780 - va
l_loss: 0.1506 - val_accuracy: 0.9451
Epoch 21/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0951 - accuracy: 0.9808 - va
l_loss: 0.1481 - val_accuracy: 0.9451
Epoch 22/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0914 - accuracy: 0.9808 - va
l_loss: 0.1455 - val_accuracy: 0.9560
Epoch 23/30
12/12 [==============================] - 0s 7ms/step - loss: 0.0878 - accuracy: 0.9808 - va
l_loss: 0.1431 - val_accuracy: 0.9560
Epoch 24/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0847 - accuracy: 0.9808 - va
l_loss: 0.1415 - val_accuracy: 0.9560
Epoch 25/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0817 - accuracy: 0.9808 - va
l_loss: 0.1397 - val_accuracy: 0.9560
Epoch 26/30
12/12 [==============================] - 0s 7ms/step - loss: 0.0791 - accuracy: 0.9890 - va
l_loss: 0.1388 - val_accuracy: 0.9560
Epoch 27/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0765 - accuracy: 0.9890 - va
l_loss: 0.1364 - val_accuracy: 0.9560
Epoch 28/30
12/12 [==============================] - 0s 7ms/step - loss: 0.0742 - accuracy: 0.9890 - va
l_loss: 0.1349 - val_accuracy: 0.9560
```

```
l_loss: 0.1349 - val_accuracy: 0.9560
Epoch 29/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0719 - accuracy: 0.9890 - va
l_loss: 0.1340 - val_accuracy: 0.9560
Epoch 30/30
12/12 [==============================] - 0s 8ms/step - loss: 0.0698 - accuracy: 0.9890 - va
l_loss: 0.1325 - val_accuracy: 0.9560
```

In [ ]:

```
model.evaluate(X_test_norm, y_test)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.1212 - accuracy: 0.9474
```

Out[ ]:

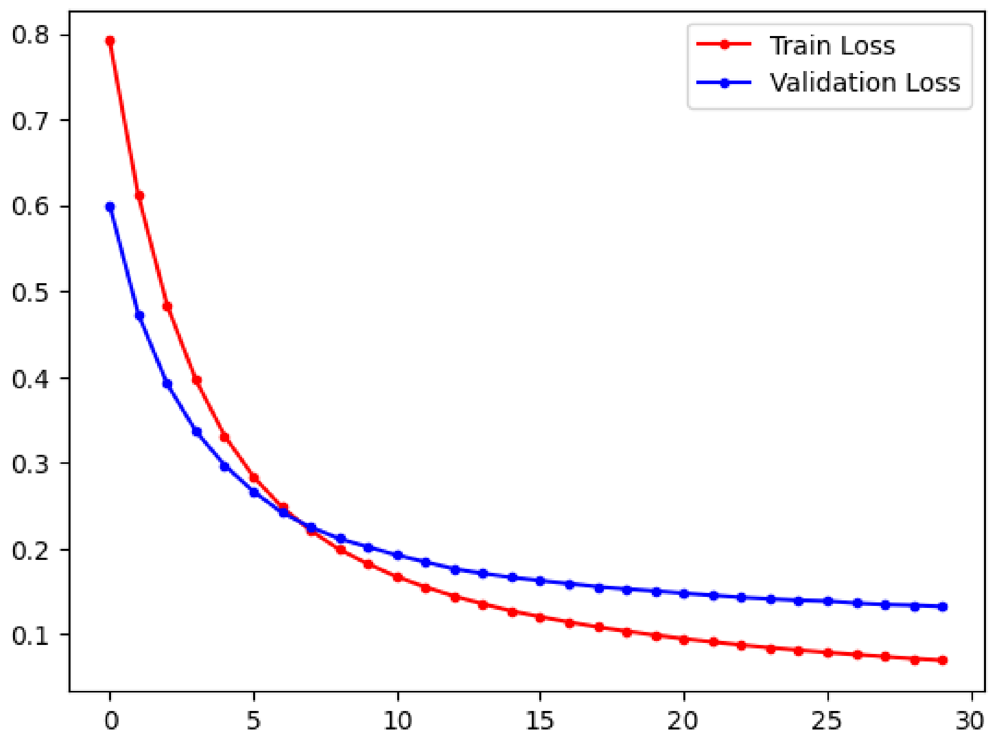
```
[0.12123900651931763, 0.9473684430122375]
```

In [ ]:

```
fig, ax = plt.subplots()
ax.plot(bCancer_model1.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(bCancer_model1.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7c32d836ebf0>
```

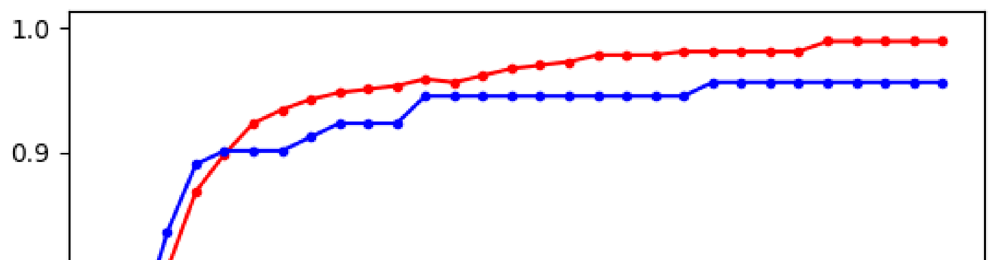


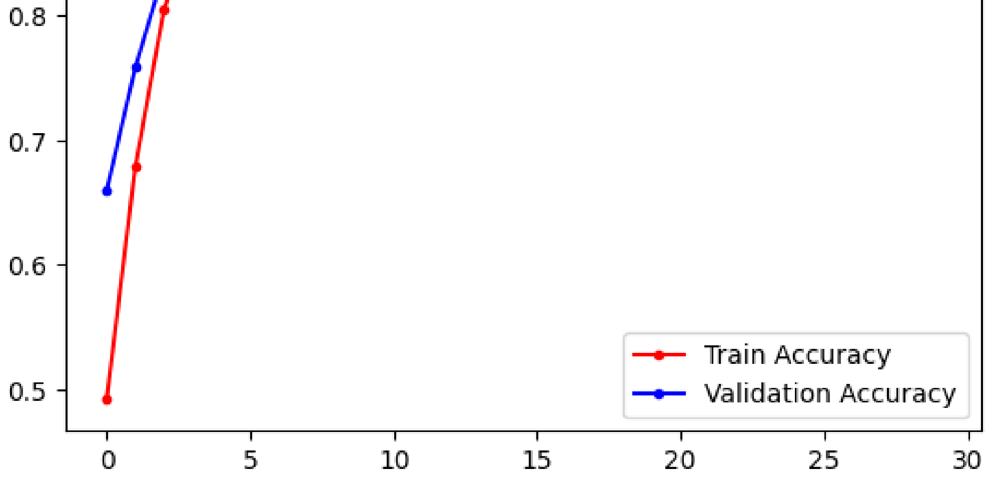
In [ ]:

```
fig, ax = plt.subplots()
ax.plot(bCancer_model1.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(bCancer_model1.history["val_accuracy"],'b', marker='.', label="Validation Accuracy
")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7c32d0be1f90>
```





Observation:

- Here, we can see that our base model has a pretty high accuracy and low loss level. It has a training accuracy of (0.9890) and training loss of (0.0698), and its validation accuracy is (0.9560) with the validation loss of (0.1325). We can see that from the accuracy of training and validation, this model performs very well and it has an accurate prediction for the validation set.
- I tested the model with the testing set and it gave a result of (0.9473) accuracy and (0.1212) loss, this further shows that this model performs very well and there is just a slight overfitting that is evident.
- We can also observe the graphs aboves, which compares the accuracy and loss of the training and validation set. We can observe from the graph that the results of the accuracy and loss of training and validation set are pretty close to each other. In the loss graph, we can observe that the line for training loss and validation loss both follow a downward slope, and the value of validation loss became stagnant around slightly below 0.2 loss and 20th epoch. For the accuracy, both training and validation follows an upward slope, and the validation accuracy became stagnant above 0.9, and it increased in the 19th epoch.

Using k Fold cross validation to test the base model

In [ ]:

```
estimators = [('standardize', StandardScaler()),
             ('mlp', KerasClassifier(model = cl_baseline_model, epochs=30, verbose = 0))]

pipeline = Pipeline(estimators)

kfold = KFold(n_splits=10)

results0 = cross_val_score(pipeline, X = X, y = y, cv=kfold, scoring='accuracy')
print("Baseline: %.4f (%.4f) Accuracy" % (results0.mean(), results0.std()))
```

```
Baseline: 0.9693 (0.0243) Accuracy
```

In [ ]:

```
print(results0)
```

```
[1.         0.91304348 1.         1.         0.97826087 0.97777778
 0.97777778 0.93333333 0.97777778 0.97777778]
```

Observation:

- Here, I used KFold cross validation to further test the data against different folds or different sets of new data. We can see that the baseline accuracy is 0.9692 with std of 0.0201. We can also see the raw accuracy for every fold that the cross_val_score returns, the motel performs very well and the lowest accuracy is 0.9130.

Optimizing the model by adding hidden layers and hidden nodes

- **Here, I added another hidden layer with 13 hidden layer nodes, which is 2/3 of the previous 20 hidden layer nodes [3]. I also added a kernel initializer for the weight and also added dropout layers to prevent overfitting.**
- **The dropout technique is a regularization technique that reduces the number of nodes randomly which reduces overfitting and co-adapting [5].**

In [ ]:

```
def cl_optimized_model():
  model1 = tf.keras.models.Sequential([
      tf.keras.layers.Dense(20, input_shape=(30,),
                             kernel_initializer="normal", activation = "relu"),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(13,
                             kernel_initializer="normal",activation = "relu"),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(1,
                             kernel_initializer="normal", activation = "sigmoid")

  ])

  model1.compile(Adam(learning_rate = 0.001),
                loss = "binary_crossentropy",
                metrics=["accuracy"]
                )
  return model1
clmodel1 = cl_optimized_model()

bCancer_model2 = clmodel1.fit(X_train_norm, y_train, validation_data = (X_val_norm, y_val)
,epochs = 30)
```

```
Epoch 1/30
12/12 [==============================] - 1s 29ms/step - loss: 0.6895 - accuracy: 0.7308 - v
al_loss: 0.6850 - val_accuracy: 0.8132
Epoch 2/30
12/12 [==============================] - 0s 8ms/step - loss: 0.6770 - accuracy: 0.8516 - va
l_loss: 0.6695 - val_accuracy: 0.8791
Epoch 3/30
12/12 [==============================] - 0s 8ms/step - loss: 0.6549 - accuracy: 0.9066 - va
l_loss: 0.6402 - val_accuracy: 0.9341
Epoch 4/30
12/12 [==============================] - 0s 6ms/step - loss: 0.6123 - accuracy: 0.9341 - va
l_loss: 0.5879 - val_accuracy: 0.9560
Epoch 5/30
12/12 [==============================] - 0s 6ms/step - loss: 0.5401 - accuracy: 0.9505 - va
l_loss: 0.5054 - val_accuracy: 0.9560
Epoch 6/30
12/12 [==============================] - 0s 8ms/step - loss: 0.4458 - accuracy: 0.9478 - va
l_loss: 0.4074 - val_accuracy: 0.9560
Epoch 7/30
12/12 [==============================] - 0s 7ms/step - loss: 0.3492 - accuracy: 0.9533 - va
l_loss: 0.3187 - val_accuracy: 0.9560
Epoch 8/30
12/12 [==============================] - 0s 8ms/step - loss: 0.2604 - accuracy: 0.9533 - va
l_loss: 0.2537 - val_accuracy: 0.9560
Epoch 9/30
12/12 [==============================] - 0s 8ms/step - loss: 0.2103 - accuracy: 0.9533 - va
l_loss: 0.2087 - val_accuracy: 0.9560
Epoch 10/30
12/12 [==============================] - 0s 8ms/step - loss: 0.1723 - accuracy: 0.9588 - va
l_loss: 0.1796 - val_accuracy: 0.9560
Epoch 11/30
12/12 [==============================] - 0s 8ms/step - loss: 0.1453 - accuracy: 0.9643 - va
l_loss: 0.1586 - val_accuracy: 0.9670
Epoch 12/30
12/12 [==============================] - 0s 9ms/step - loss: 0.1293 - accuracy: 0.9670 - va
l_loss: 0.1453 - val_accuracy: 0.9670
```

```
l_loss: 0.1453 - val_accuracy: 0.9670
Epoch 13/30
12/12 [==============================] - 0s 7ms/step - loss: 0.1097 - accuracy: 0.9698 - va
l_loss: 0.1379 - val_accuracy: 0.9560
Epoch 14/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0996 - accuracy: 0.9780 - va
l_loss: 0.1332 - val_accuracy: 0.9560
Epoch 15/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0946 - accuracy: 0.9808 - va
l_loss: 0.1300 - val_accuracy: 0.9560
Epoch 16/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0830 - accuracy: 0.9863 - va
l_loss: 0.1275 - val_accuracy: 0.9670
Epoch 17/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0800 - accuracy: 0.9835 - va
l_loss: 0.1254 - val_accuracy: 0.9670
Epoch 18/30
12/12 [==============================] - 0s 7ms/step - loss: 0.0719 - accuracy: 0.9835 - va
l_loss: 0.1253 - val_accuracy: 0.9560
Epoch 19/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0662 - accuracy: 0.9835 - va
l_loss: 0.1236 - val_accuracy: 0.9560
Epoch 20/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0722 - accuracy: 0.9808 - va
l_loss: 0.1200 - val_accuracy: 0.9670
Epoch 21/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0658 - accuracy: 0.9863 - va
l_loss: 0.1182 - val_accuracy: 0.9670
Epoch 22/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0726 - accuracy: 0.9863 - va
l_loss: 0.1176 - val_accuracy: 0.9670
Epoch 23/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0591 - accuracy: 0.9835 - va
l_loss: 0.1186 - val_accuracy: 0.9670
Epoch 24/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0619 - accuracy: 0.9890 - va
l_loss: 0.1209 - val_accuracy: 0.9560
Epoch 25/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0630 - accuracy: 0.9863 - va
l_loss: 0.1237 - val_accuracy: 0.9451
Epoch 26/30
12/12 [==============================] - 0s 7ms/step - loss: 0.0612 - accuracy: 0.9808 - va
l_loss: 0.1239 - val_accuracy: 0.9451
Epoch 27/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0555 - accuracy: 0.9918 - va
l_loss: 0.1235 - val_accuracy: 0.9560
Epoch 28/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0528 - accuracy: 0.9863 - va
l_loss: 0.1246 - val_accuracy: 0.9451
Epoch 29/30
12/12 [==============================] - 0s 6ms/step - loss: 0.0547 - accuracy: 0.9863 - va
l_loss: 0.1242 - val_accuracy: 0.9560
Epoch 30/30
12/12 [==============================] - 0s 5ms/step - loss: 0.0502 - accuracy: 0.9918 - va
l_loss: 0.1266 - val_accuracy: 0.9560
```

In [ ]:

```
clmodel1.evaluate(X_test_norm, y_test)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.1067 - accuracy: 0.9649
```

Out[ ]:

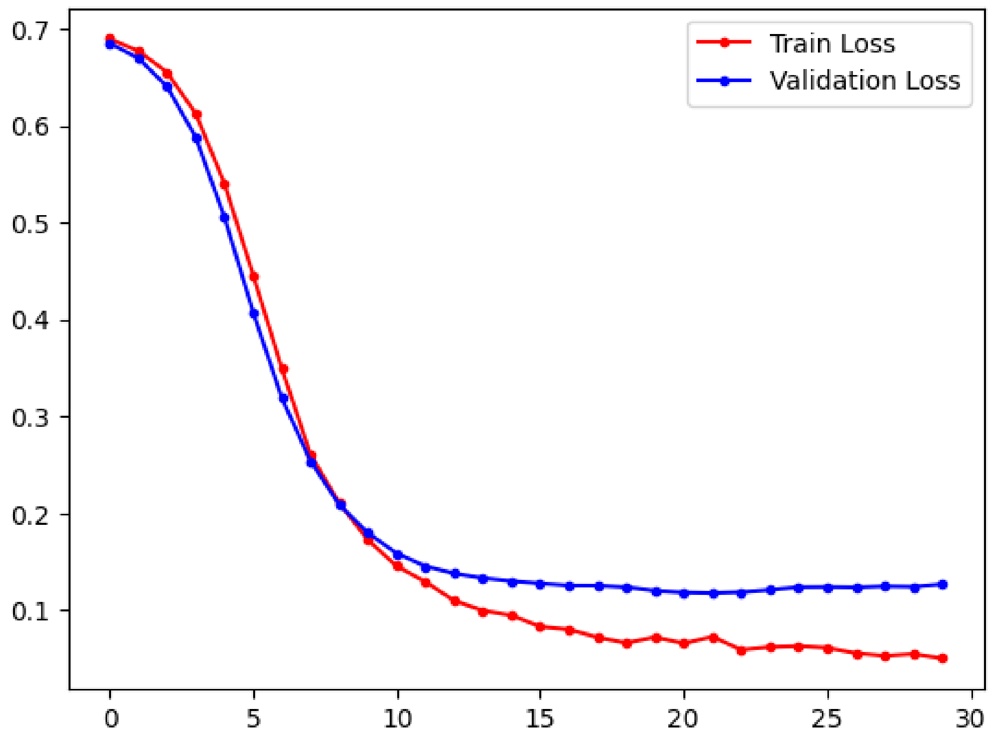
```
[0.10665810108184814, 0.9649122953414917]
```

In [ ]:

```
fig, ax = plt.subplots()
ax.plot(bCancer_model2.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(bCancer_model2.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7c32c78a0970>
```

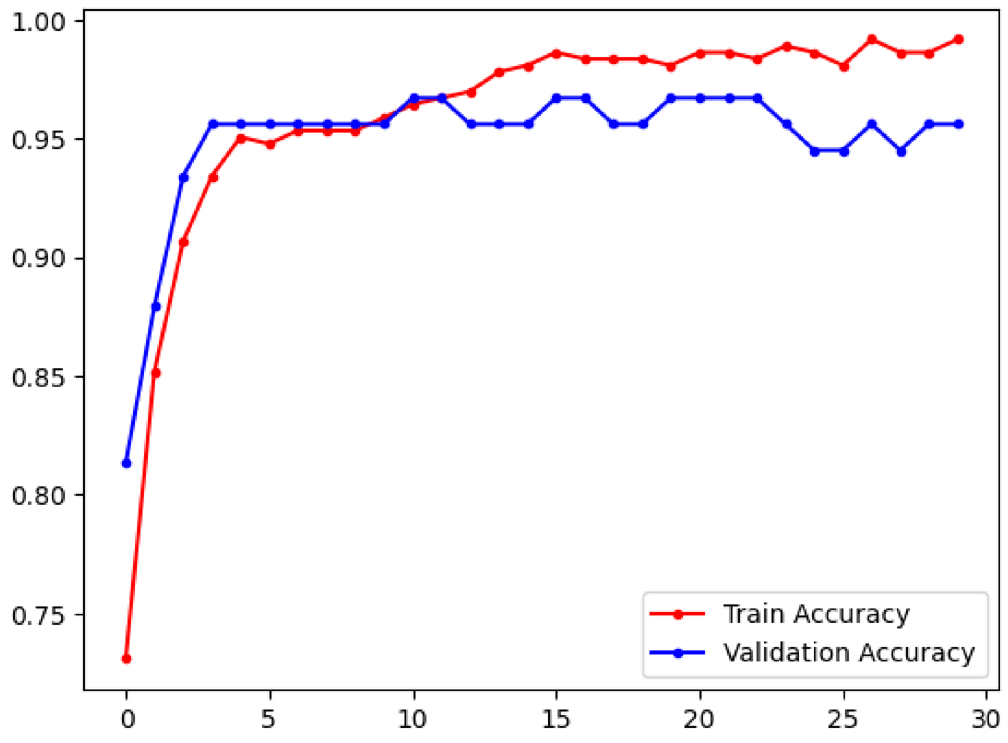


In [ ]:

```
fig, ax = plt.subplots()
ax.plot(bCancer_model2.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(bCancer_model2.history["val_accuracy"],'b', marker='.', label="Validation Accuracy
")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7c32b97d2920>
```



In [ ]:

```
estimators = [('standardize', StandardScaler()),
              ('mlp', KerasClassifier(model = cl_optimized_model, epochs=30, verbose = 0))]

pipeline = Pipeline(estimators)
```

```
kfold = KFold(n_splits=10)

results1 = cross_val_score(pipeline, X = X, y = y, cv=kfold, scoring='accuracy')
print("Optimized: %.4f (%.4f) Accuracy" % (results1.mean(), results1.std()))
```

```
Optimized: 0.9780 (0.0241) Accuracy
```

In [ ]:

```
print(results1)
```

```
[1.         0.93478261 1.         0.97826087 0.97826087 1.
 0.97777778 0.95555556 1.         0.95555556]
```

Observation:

- **Here in this optimized model of our classification dataset, I increased the hidden layer to 2 and the 2nd hidden layer has 13 hidden layer nodes. I also added dropout layers to reduce overfitting and to train a better model. For the other parameters, I did not change anything and kept them the same as the base model.**
- **We can observe that the accuracy for the training and validation set of the model is much lower than the previous base model. The training set of this model has (0.9918) accuracy and (0.0502) loss, while the validation set has (0.9560) accuracy and (0.1266) loss. The validation accuracy is much higher than the training accuracy which means that this model is not overfitted.**
- **I tested this optimized model to the testing data and found out that this model has a higher accuracy and lower loss compared to the previous base model. This model has a testing accuracy of (0.9649) and testing loss of (0.1066). This shows that adding hidden layer and hidden layer nodes further optimized the predictive power of this model.**
- **We can observe the graphs above, it shows a much better slope than the previous graphs of the base model. For the loss graph of training and validation set, we can see that the loss of this model became much lower than the previous model and it did not became stagnant in the 20th epoch. For the accuracy graph of this model, we can see that it is even much higher than the training accuracy. It also did not became stagnant and instead reached a higher accuracy.**
- **I also used the KFold cross validation to further test this model, and as you can see in the result, the accuracy is 0.9780 with an std of 0.0201. You can also see in the printed results the raw values returned by the cross_val_score method.**
- **Overall, I was successful in utilzing hidden layers and hidden layer nodes to optimize the base model. I achieved a higher accuracy and lower loss in the optimized model than the base model.**

REGRESSION TASK:

Performing Exploratory Data Analysis on Regression Dataset

- **Let's perform Exploratory Data Analysis on Regression Dataset.**

In [ ]:

```
mpg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   car_name      398 non-null    object 
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    int64
 8   mpg           398 non-null    float64
dtypes: float64(4), int64(4), object(1)
memory usage: 28.1+ KB
```

Observation:

- **Here we can observe that there are 9 features from which 7 are predictors and 1 is the target variable. We can see that there are 4 float64 data type, 4 int64 data type, and 1 object data type. The object is the car name which is a uniquie identifier for each entry. The target variable here is MPG or Mile per gallon.**

In [ ]:

```
mpg_df.isnull().sum()
```

Out[ ]:

```
car_name        0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
mpg             0
dtype: int64
```

Observation:

- **Here, we can observe that there are 6 missing or null values in the column of horsepower. We can proceed by filling the horsepower column with the mean of the values of the column.**

In [6]:

```
mpg_df["horsepower"] = mpg_df["horsepower"].fillna(mpg_df["horsepower"].mean())
mpg_df.isnull().sum()
```

Out[6]:

```
car_name        0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
mpg             0
dtype: int64
```

In [ ]:

```
mpg_df.head()
```

Out[ ]:

	car_name	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	mpg
0	chevrolet,chevelle,malibu	8	307.0	130.0	3504	12.0	70	1	18.0
1	buick,skylark,320	8	350.0	165.0	3693	11.5	70	1	15.0

	car_name	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	mpg
2	plymouth,satellite	8	318.0	150.0	3436	11.0	70	1	18.0
3	amc,rebel,sst	8	304.0	150.0	3433	12.0	70	1	16.0
4	ford,torino	8	302.0	140.0	3449	10.5	70	1	17.0

In [ ]:

```
mpg_df.describe()
```

Out[ ]:

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	mpg
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864	23.514573
std	1.701004	104.269838	38.199187	846.841774	2.757689	3.697627	0.802055	7.815984
min	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000	9.000000
25%	4.000000	104.250000	76.000000	2223.750000	13.825000	73.000000	1.000000	17.500000
50%	4.000000	148.500000	95.000000	2803.500000	15.500000	76.000000	1.000000	23.000000
75%	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000	2.000000	29.000000
max	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000	46.600000

- **To proceed with the correlation, I will be dropping the column "car_name" as it is only a unique identifier and it can affect the execution of the heatmap.**

In [7]:

```
mpg_df_1 = mpg_df.drop(["car_name"], axis = 1)
mpg_df_1.head()
```

Out[7]:

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	mpg
0	8	307.0	130.0	3504	12.0	70	1	18.0
1	8	350.0	165.0	3693	11.5	70	1	15.0
2	8	318.0	150.0	3436	11.0	70	1	18.0
3	8	304.0	150.0	3433	12.0	70	1	16.0
4	8	302.0	140.0	3449	10.5	70	1	17.0

In [ ]:

```
mpg_df_1.corr(method = "pearson" )
```

Out[ ]:

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	mpg
cylinders	1.000000	0.950721	0.838939	0.896017	-0.505419	-0.348746	-0.562543	-0.775396
displacement	0.950721	1.000000	0.893646	0.932824	-0.543684	-0.370164	-0.609409	-0.804203
horsepower	0.838939	0.893646	1.000000	0.860574	-0.684259	-0.411651	-0.453669	-0.771437
weight	0.896017	0.932824	0.860574	1.000000	-0.417457	-0.306564	-0.581024	-0.831741
acceleration	-0.505419	-0.543684	-0.684259	-0.417457	1.000000	0.288137	0.205873	0.420289
model_year	-0.348746	-0.370164	-0.411651	-0.306564	0.288137	1.000000	0.180662	0.579267
origin	-0.562543	-0.609409	-0.453669	-0.581024	0.205873	0.180662	1.000000	0.563450
mpg	-0.775396	-0.804203	-0.771437	-0.831741	0.420289	0.579267	0.563450	1.000000

In [ ]:

```
mpg_df_1.corr(method = "pearson")["mpg"].sort_values(ascending = False)
```

Out[ ]:

```
mpg             1.000000
model_year      0.579267
origin          0.563450
acceleration    0.420289
horsepower     -0.771437
cylinders      -0.775396
displacement   -0.804203
weight         -0.831741
Name: mpg, dtype: float64
```

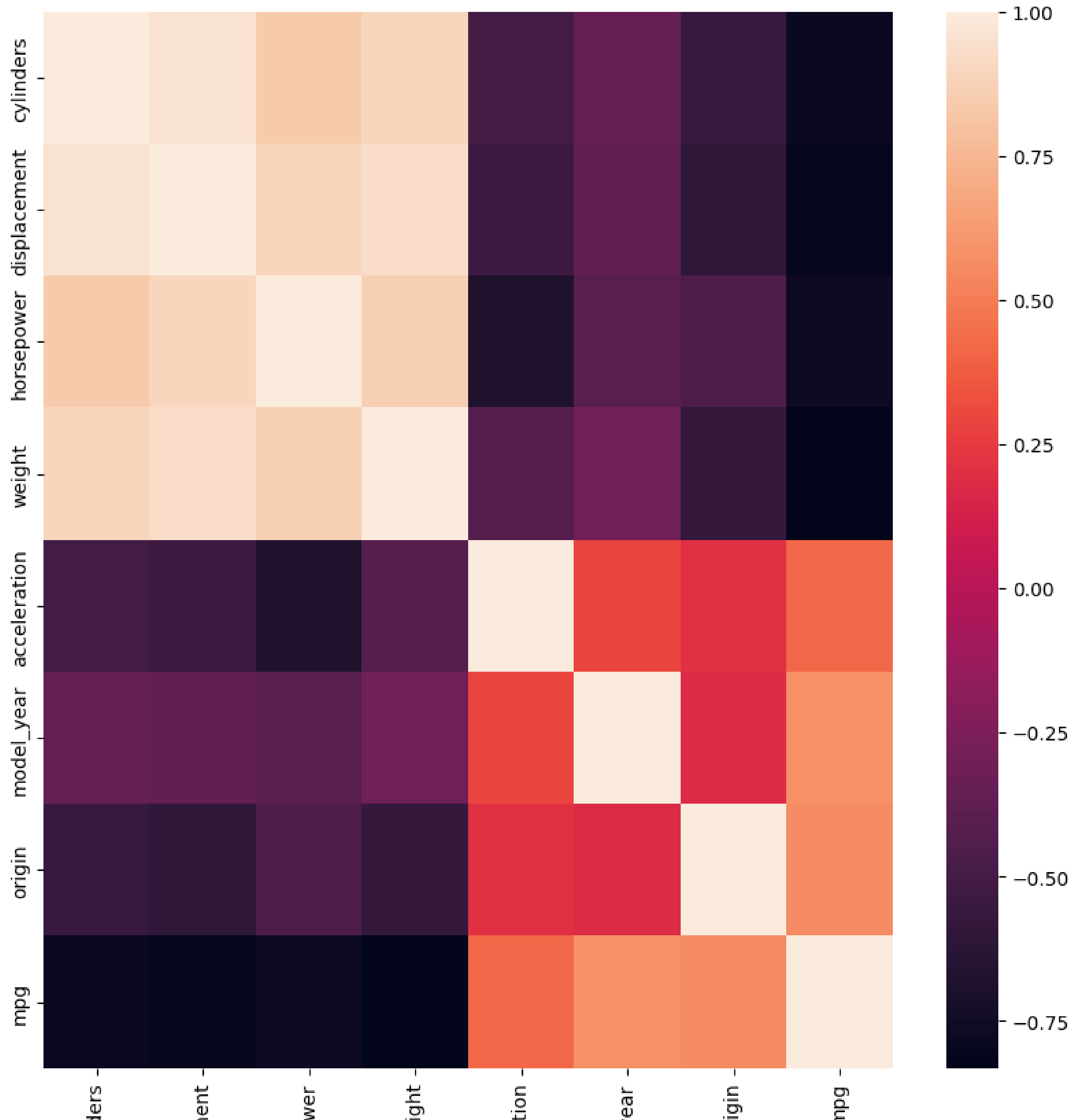
In [ ]:

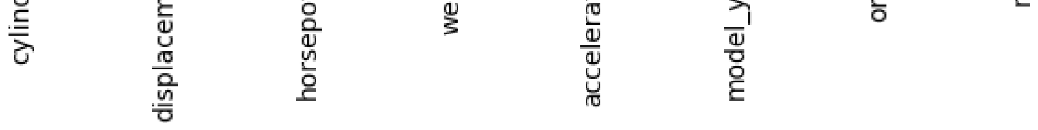
```
fig1 = plt.figure(figsize=(10,10), dpi=100)

sns.heatmap(mpg_df_1.corr())
```

Out[ ]:

```
<Axes: >
```





Observation:

- From the heatmap and correlational values above, we can observe that there are variables which has a moderate correlation with the target variable "mpg". This indicates that there are variables with relationship with our mpg that can be used later to improve the performance of our model.

Preparing and Splitting the dataset into Training, Validation and Testing sets (Regression)

- Let's split the dataset into training, validation and testing sets. The split will be 60/20/20, 60 for training, 20 for validation and 20 for testing [1,2].
- I will be standardizing the dataset now but for the base model, we will be using not standardized dataset.

In [8]:

```
X1 = mpg_df_1.drop(["mpg"], axis = 1)
y1 = mpg_df_1["mpg"]

X1, X1_test, y1, y1_test = train_test_split(X1, y1, test_size =0.2, random_state=100)
X1_train, X1_val, y1_train, y1_val = train_test_split(X1, y1, test_size =0.2, random_state
=110)
```

In [9]:

```
scaler1 = StandardScaler()
X1_train_norm = scaler1.fit_transform(X1_train)
X1_val_norm = scaler1.transform(X1_val)
X1_test_norm = scaler1.transform(X1_test)
```

Training the Base Model for Regression

- In training the base model of this dataset, I will be using the original dataset and not the normalized dataset. For the first hidden layer, I will be using the same number of input nodes which is 7 and I will use 1 output node.
- For the activation of the hidden layer, I used relu while for the output layer I used linear activation that is usually used for regression model.

In [10]:

```
def r_square(y_true, y_pred):
    from keras import backend as K
    SSres =  K.sum(K.square(y_true - y_pred))
    SStot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SSres/(SStot + K.epsilon()))
```

In [26]:

```
def reg_baseline_model():

  regmodel = tf.keras.models.Sequential([
      tf.keras.layers.Dense(7, input_shape=(7,), kernel_initializer = "normal",
                            activation = "relu"),
      tf.keras.layers.Dense(1,kernel_initializer = "normal",activation = "linear")
  ])
  regmodel.compile(optimizer = Adam(learning_rate = 0.1),
                   loss = "mean_squared_error",
                   metrics=[r_square]
                   )
  return regmodel

baseline_model = reg_baseline_model()

mpg_df_1_model1 = baseline_model.fit(X1_train, y1_train,
                                     validation_data = (X1_val, y1_val),
                                     epochs = 30)
```

```
Epoch 1/30
8/8 [==============================] - 1s 49ms/step - loss: 637.1766 - r_square: -9.8609 -
val_loss: 602.7711 - val_r_square: -8.2496
Epoch 2/30
8/8 [==============================] - 0s 13ms/step - loss: 567.0647 - r_square: -8.8455 -
val_loss: 566.9135 - val_r_square: -7.6973
Epoch 3/30
8/8 [==============================] - 0s 12ms/step - loss: 532.4462 - r_square: -8.3472 -
val_loss: 532.6747 - val_r_square: -7.1700
Epoch 4/30
8/8 [==============================] - 0s 13ms/step - loss: 499.1284 - r_square: -7.4352 -
val_loss: 500.1003 - val_r_square: -6.6684
Epoch 5/30
8/8 [==============================] - 0s 11ms/step - loss: 467.8313 - r_square: -6.9904 -
val_loss: 469.0427 - val_r_square: -6.1903
Epoch 6/30
8/8 [==============================] - 0s 11ms/step - loss: 438.0919 - r_square: -6.4884 -
val_loss: 439.6036 - val_r_square: -5.7371
Epoch 7/30
8/8 [==============================] - 0s 12ms/step - loss: 409.9058 - r_square: -6.0015 -
val_loss: 411.7971 - val_r_square: -5.3092
Epoch 8/30
8/8 [==============================] - 0s 35ms/step - loss: 383.3210 - r_square: -5.5944 -
val_loss: 385.6320 - val_r_square: -4.9066
Epoch 9/30
8/8 [==============================] - 0s 28ms/step - loss: 358.0673 - r_square: -5.2049 -
val_loss: 361.1663 - val_r_square: -4.5302
Epoch 10/30
8/8 [==============================] - 0s 12ms/step - loss: 334.8930 - r_square: -4.7817 -
val_loss: 337.9615 - val_r_square: -4.1733
Epoch 11/30
8/8 [==============================] - 0s 17ms/step - loss: 312.5712 - r_square: -4.6849 -
val_loss: 316.3927 - val_r_square: -3.8417
Epoch 12/30
8/8 [==============================] - 0s 31ms/step - loss: 291.9767 - r_square: -4.0807 -
val_loss: 296.1514 - val_r_square: -3.5305
Epoch 13/30
8/8 [==============================] - 0s 6ms/step - loss: 272.6838 - r_square: -3.7486 - v
al_loss: 277.2291 - val_r_square: -3.2397
Epoch 14/30
8/8 [==============================] - 0s 9ms/step - loss: 254.7294 - r_square: -3.3323 - v
al_loss: 259.5380 - val_r_square: -2.9679
Epoch 15/30
8/8 [==============================] - 0s 6ms/step - loss: 238.0203 - r_square: -3.1616 - v
al_loss: 243.0199 - val_r_square: -2.7142
Epoch 16/30
8/8 [==============================] - 0s 9ms/step - loss: 222.4310 - r_square: -2.7446 - v
al_loss: 227.6467 - val_r_square: -2.4782
Epoch 17/30
8/8 [==============================] - 0s 8ms/step - loss: 207.8609 - r_square: -2.5841 - v
al_loss: 213.4525 - val_r_square: -2.2603
Epoch 18/30
8/8 [==============================] - 0s 6ms/step - loss: 194.6625 - r_square: -2.3024 - v
al_loss: 200.1545 - val_r_square: -2.0562
Epoch 19/30
8/8 [==============================] - 0s 6ms/step - loss: 182.0016 - r_square: -2.1236 - v
al_loss: 188.0412 - val_r_square: -1.8705
Epoch 20/30
8/8 [==============================] - 0s 6ms/step - loss: 170.6346 - r_square: -2.0210 - v
al_loss: 176.7953 - val_r_square: -1.6980
```

```
Epoch 21/30
8/8 [==============================] - 0s 9ms/step - loss: 160.0667 - r_square: -1.8162 - v
al_loss: 166.4492 - val_r_square: -1.5395
Epoch 22/30
8/8 [==============================] - 0s 8ms/step - loss: 150.3833 - r_square: -1.6027 - v
al_loss: 156.9244 - val_r_square: -1.3936
Epoch 23/30
8/8 [==============================] - 0s 7ms/step - loss: 141.3571 - r_square: -1.4263 - v
al_loss: 148.2714 - val_r_square: -1.2611
Epoch 24/30
8/8 [==============================] - 0s 7ms/step - loss: 133.5714 - r_square: -1.2387 - v
al_loss: 140.0703 - val_r_square: -1.1356
Epoch 25/30
8/8 [==============================] - 0s 9ms/step - loss: 126.0124 - r_square: -1.1826 - v
al_loss: 132.6515 - val_r_square: -1.0222
Epoch 26/30
8/8 [==============================] - 0s 6ms/step - loss: 118.9811 - r_square: -1.1243 - v
al_loss: 126.0611 - val_r_square: -0.9215
Epoch 27/30
8/8 [==============================] - 0s 9ms/step - loss: 113.0223 - r_square: -0.9164 - v
al_loss: 119.8676 - val_r_square: -0.8269
Epoch 28/30
8/8 [==============================] - 0s 6ms/step - loss: 107.2311 - r_square: -0.8176 - v
al_loss: 114.3631 - val_r_square: -0.7429
Epoch 29/30
8/8 [==============================] - 0s 7ms/step - loss: 102.2212 - r_square: -0.7256 - v
al_loss: 109.3039 - val_r_square: -0.6657
Epoch 30/30
8/8 [==============================] - 0s 9ms/step - loss: 97.6719 - r_square: -0.6482 - va
l_loss: 104.6784 - val_r_square: -0.5952
```

In [ ]:

```
baseline_model.evaluate(X1_test, y1_test)
```

```
3/3 [==============================] - 0s 9ms/step - loss: 84.6856 - r_square: -0.5804
```

Out[ ]:

```
[84.68557739257812, -0.5804415345191956]
```

In [25]:

```
#evaluate model
estimators = [('mlp', KerasRegressor(model = baseline_model, epochs=30, verbose = 0))]
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X = X1, y = y1, cv=kfold, scoring='neg_mean_squared_er
ror')
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Baseline: -65.87 (16.91) MSE
```

In [ ]:

```
y_pred = baseline_model.predict(X1_test)
```

```
3/3 [==============================] - 0s 6ms/step
```

In [ ]:

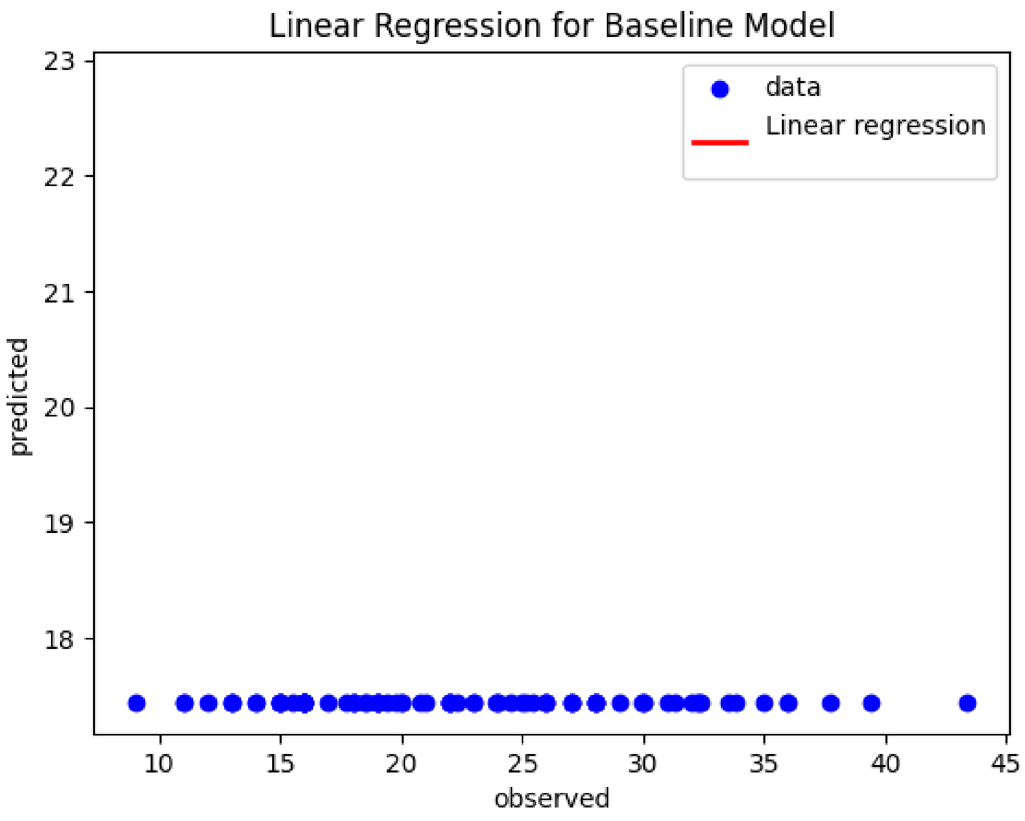
```
from sklearn.linear_model import LinearRegression

def plot_regression(pred):
  regressor = LinearRegression()
  y_pred_reshaped = pred.reshape(-1, 1)
  regressor.fit(y_pred_reshaped, y1_test)
  y_fit = regressor.predict(y_pred_reshaped)
  reg_intercept = round(regressor.intercept_, 4)
  reg_coef = round(regressor.coef_.flatten()[0], 4)
  return y_fit

newy_fit = plot_regression(y_pred)
```

In [ ]:

```
plt.scatter(y1_test, y_pred, color='blue', label= 'data')
plt.plot(y_pred, newy_fit, color='red', linewidth=2, label = 'Linear regression\n')
plt.title('Linear Regression for Baseline Model')
plt.legend()
plt.xlabel('observed')
plt.ylabel('predicted')
plt.show()
```



Observation:

- We can observe above that the result of the loss for this dataset is high, while the r^2 also became negative. The final loss and r^2 for training is (97.6719) and (-0.6482) respectively. For the validation set, the loss is (104.6784) and the r_square is (-0.5952).
- I also tested this baseline model using the testing dataset and the outcome is that the loss is (84.6855) and the r^2 is (-0.580). The loss decreased for the testing dataset while the r^2 is still negative.
- Our loss is indicated by mean squared error, this metric is used to measure the value of the error that our model makes. For the r^2, this metric is used to represent how well the regression line fits the dataset. The closer it is to 1, the better our model is. In this case, the r^2 is negative so this means that this model is a really bad fit for our data. We can see the visualization of this in the plotted linear regression above. In the plot above, there is no line, this is because this model can't predict the data points present.
- For the kfold cross validation, we can see the result of it which is -65.87 MSE. This is a negative mean squared error, which is a scoring metric for a regression model that indicates that the higher the value the more the model is well fitted. In our case, -65.87 is a very low value and it is not even close to 0, when the value is closer to 0, it indicates a good result.
- This baseline model is a bad fit for our dataset, mainly because the dataset is not normalized yet.

Optimizing the model by standardizing the dataset

- For this model, I used the standardized X_train, X_val and X_test to train and test this model.

In [ ]:

```
def optimized_model():
  regmodel1 = tf.keras.models.Sequential([
      tf.keras.layers.Dense(7, input_shape=(7,), kernel_initializer = "normal",
                            activation = "relu"),
      tf.keras.layers.Dense(1,  kernel_initializer = "normal",
                            activation = "linear")
  ])

  regmodel1.compile(Adam(learning_rate = 0.1),
                 loss = "mean_squared_error",
                 metrics=[r_square]
                 )
  return regmodel1

new_model = optimized_model()

mpg_df_1_model2 = new_model.fit(X1_train_norm, y1_train,
                                validation_data = (X1_val_norm, y1_val),
                                epochs = 30)
```

```
Epoch 1/30
1/8 [==>...........................] - ETA: 4s - loss: 570.9824 - r_square: -7.6501
```

```
WARNING:tensorflow:5 out of the last 70 calls to <function Model.make_test_function.<locals
>.test_function at 0x7c32a6e71750> triggered tf.function retracing. Tracing is expensive an
d the excessive number of tracings could be due to (1) creating @tf.function repeatedly in
a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of te
nsors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refe
r to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensor
flow.org/api_docs/python/tf/function for  more details.
```

```
8/8 [==============================] - 1s 27ms/step - loss: 516.8290 - r_square: -8.0641 -
val_loss: 346.7441 - val_r_square: -4.3148
Epoch 2/30
8/8 [==============================] - 0s 6ms/step - loss: 203.0979 - r_square: -2.5924 - v
al_loss: 60.0164 - val_r_square: 0.0812
Epoch 3/30
8/8 [==============================] - 0s 6ms/step - loss: 76.1331 - r_square: -0.3924 - va
l_loss: 90.5387 - val_r_square: -0.3823
Epoch 4/30
8/8 [==============================] - 0s 8ms/step - loss: 44.0680 - r_square: 0.2011 - val
_loss: 34.5275 - val_r_square: 0.4724
Epoch 5/30
8/8 [==============================] - 0s 9ms/step - loss: 28.0915 - r_square: 0.5282 - val
_loss: 32.4773 - val_r_square: 0.5066
Epoch 6/30
8/8 [==============================] - 0s 6ms/step - loss: 17.2443 - r_square: 0.6768 - val
_loss: 25.8823 - val_r_square: 0.6041
Epoch 7/30
8/8 [==============================] - 0s 6ms/step - loss: 17.1347 - r_square: 0.6686 - val
_loss: 20.6397 - val_r_square: 0.6871
Epoch 8/30
8/8 [==============================] - 0s 8ms/step - loss: 13.9194 - r_square: 0.7679 - val
_loss: 18.7964 - val_r_square: 0.7149
Epoch 9/30
8/8 [==============================] - 0s 8ms/step - loss: 12.0344 - r_square: 0.7917 - val
_loss: 15.3425 - val_r_square: 0.7643
Epoch 10/30
8/8 [==============================] - 0s 8ms/step - loss: 11.2114 - r_square: 0.7940 - val
_loss: 14.6066 - val_r_square: 0.7759
Epoch 11/30
8/8 [==============================] - 0s 8ms/step - loss: 10.2907 - r_square: 0.8186 - val
_loss: 13.1515 - val_r_square: 0.7981
Epoch 12/30
8/8 [==============================] - 0s 6ms/step - loss: 9.5175 - r_square: 0.8391 - val_
loss: 12.9000 - val_r_square: 0.8028
Epoch 13/30
8/8 [==============================] - 0s 6ms/step - loss: 9.1607 - r_square: 0.8496 - val_
loss: 12.8342 - val_r_square: 0.8032
Epoch 14/30
8/8 [==============================] - 0s 7ms/step - loss: 9.0508 - r_square: 0.8497 - val_
loss: 12.8098 - val_r_square: 0.8034
Epoch 15/30
```

```
8/8 [==============================] - 0s 8ms/step - loss: 8.4056 - r_square: 0.8623 - val_
loss: 12.2015 - val_r_square: 0.8131
Epoch 16/30
8/8 [==============================] - 0s 7ms/step - loss: 8.3110 - r_square: 0.8631 - val_
loss: 12.1216 - val_r_square: 0.8140
Epoch 17/30
8/8 [==============================] - 0s 9ms/step - loss: 8.2184 - r_square: 0.8595 - val_
loss: 12.6579 - val_r_square: 0.8060
Epoch 18/30
8/8 [==============================] - 0s 6ms/step - loss: 8.1287 - r_square: 0.8606 - val_
loss: 12.4351 - val_r_square: 0.8090
Epoch 19/30
8/8 [==============================] - 0s 6ms/step - loss: 8.1155 - r_square: 0.8623 - val_
loss: 12.2431 - val_r_square: 0.8115
Epoch 20/30
8/8 [==============================] - 0s 6ms/step - loss: 8.0542 - r_square: 0.8602 - val_
loss: 12.0527 - val_r_square: 0.8146
Epoch 21/30
8/8 [==============================] - 0s 8ms/step - loss: 8.0889 - r_square: 0.8626 - val_
loss: 12.4397 - val_r_square: 0.8087
Epoch 22/30
8/8 [==============================] - 0s 7ms/step - loss: 7.9635 - r_square: 0.8620 - val_
loss: 12.1352 - val_r_square: 0.8131
Epoch 23/30
8/8 [==============================] - 0s 9ms/step - loss: 7.8117 - r_square: 0.8679 - val_
loss: 11.6981 - val_r_square: 0.8196
Epoch 24/30
8/8 [==============================] - 0s 8ms/step - loss: 7.7317 - r_square: 0.8665 - val_
loss: 12.3505 - val_r_square: 0.8099
Epoch 25/30
8/8 [==============================] - 0s 6ms/step - loss: 7.9315 - r_square: 0.8656 - val_
loss: 12.4055 - val_r_square: 0.8097
Epoch 26/30
8/8 [==============================] - 0s 6ms/step - loss: 7.8443 - r_square: 0.8589 - val_
loss: 11.8418 - val_r_square: 0.8171
Epoch 27/30
8/8 [==============================] - 0s 5ms/step - loss: 7.8672 - r_square: 0.8700 - val_
loss: 12.6008 - val_r_square: 0.8061
Epoch 28/30
8/8 [==============================] - 0s 8ms/step - loss: 7.7603 - r_square: 0.8596 - val_
loss: 11.7844 - val_r_square: 0.8184
Epoch 29/30
8/8 [==============================] - 0s 8ms/step - loss: 7.8034 - r_square: 0.8680 - val_
loss: 11.6551 - val_r_square: 0.8203
Epoch 30/30
8/8 [==============================] - 0s 6ms/step - loss: 7.6299 - r_square: 0.8702 - val_
loss: 12.5962 - val_r_square: 0.8064
```

In [ ]:

```
new_model.evaluate(X1_test_norm, y1_test)
```

```
3/3 [==============================] - 0s 10ms/step - loss: 7.0560 - r_square: 0.8720
```

Out[ ]:

```
[7.056026458740234, 0.8719863295555115]
```

In [15]:

```
#evaluate model
estimators = [('standardize', StandardScaler()),
              ('mlp', KerasRegressor(model = optimized_model, epochs=30, verbose = 0))]
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X = X1, y = y1, cv=kfold, scoring='neg_mean_squared_er
ror')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Standardized: -9.25 (2.92) MSE
```

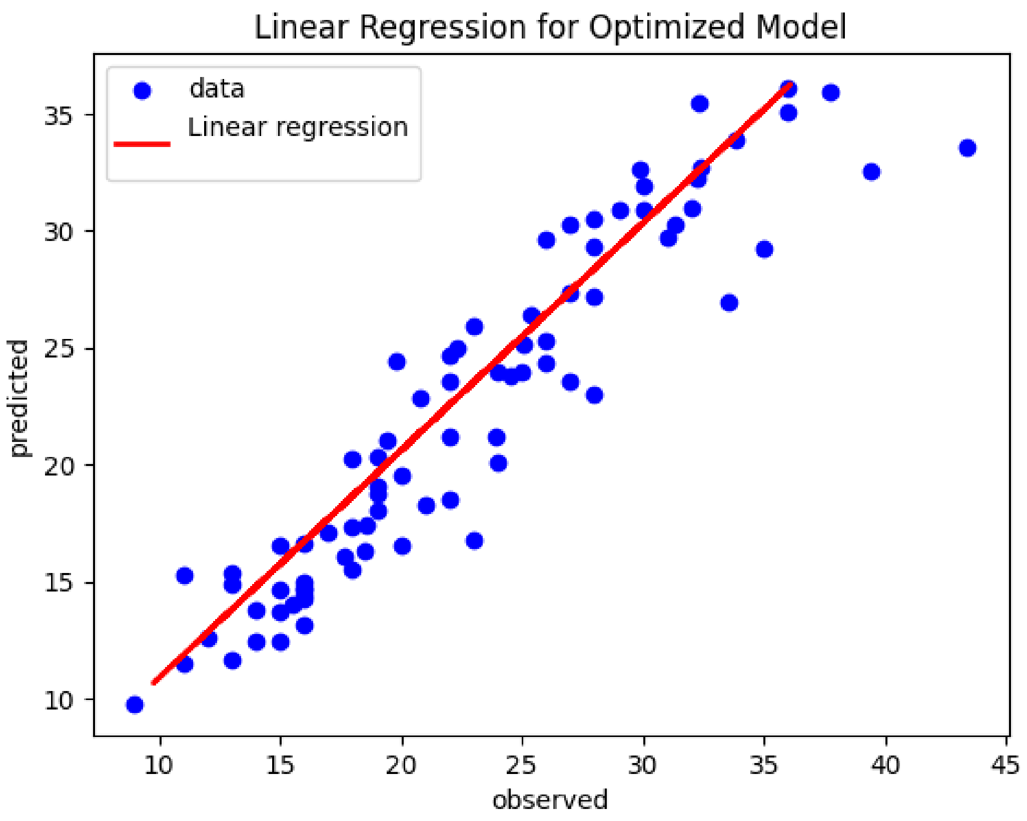
In [ ]:

```
y_pred1 = new_model.predict(X1_test_norm)
```

```
3/3 [==============================] - 0s 5ms/step
```

In [ ]:

```
optimizedy_fit = plot_regression(y_pred1)
plt.scatter(y1_test, y_pred1, color='blue', label= 'data')
plt.plot(y_pred1, optimizedy_fit, color='red', linewidth=2, label = 'Linear regression\n')
plt.title('Linear Regression for Optimized Model')
plt.legend()
plt.xlabel('observed')
plt.ylabel('predicted')
plt.show()
```



Observation:

- For this optimized model, we can observe that the loss and r^2 significantly improved. For the loss which is our mean squared error, it significantly decreased and now it only has the value of (7.6299) for training and (12.5962) for validation. For the r^2, it is no longer negative value and its value for the training is (0.8702) while for validation it is (0.8064).
- I also tested this model against the testing set and the outcome for the loss is (7.0560) while for the r^2, the result is (0.8719). This shows a significant increase in the performance of our model. The value of r^2 is now very close to 1 which means that this model is a very good fit for our dataset.
- For the kfold cross validation, the result indicates that this model is well fitted and the value of error that it makes is smaller than the baseline model. The result of MSE from the kfold cross validation is -9.25, which is closer to 0.
- We can also see the plot for visualization, we can observe that unlike the previous regression model, this plot now has a line to represent the regression line. The values of predicted and observed are also along that regression line, which means that our model predicts the values very well.
- Overall, this model is well fitted now to our dataset, as indicated by low MSE and high r^2 which is close to 1.

Tuning layers of the model (Small and Large network)

- Let's start with building our small network. I used 4 hidden layer nodes here in this network instead of 7 as

compared to the previous baseline and optimized network.

In [ ]:

```
"""Smaller network"""

def small_reg_model():

  regmodel = tf.keras.models.Sequential([
      tf.keras.layers.Dense(4, input_shape=(7,), kernel_initializer = "normal",
                            activation = "relu"),
      tf.keras.layers.Dense(1,kernel_initializer = "normal",activation = "linear")
  ])

  regmodel.compile(optimizer = Adam(learning_rate = 0.1),
                loss = "mean_squared_error",
                metrics=[r_square]
                )
  return regmodel

small_reg_network = small_reg_model()

small_reg_model_history1 = small_reg_network.fit(X1_train_norm, y1_train,
                              validation_data = (X1_val_norm, y1_val),
                              epochs = 30)
```

```
Epoch 1/30
8/8 [==============================] - 1s 51ms/step - loss: 530.7991 - r_square: -8.0874 -
val_loss: 362.8407 - val_r_square: -4.5644
Epoch 2/30
8/8 [==============================] - 0s 9ms/step - loss: 208.3844 - r_square: -2.5409 - v
al_loss: 72.6117 - val_r_square: -0.1108
Epoch 3/30
8/8 [==============================] - 0s 12ms/step - loss: 86.2952 - r_square: -0.5112 - v
al_loss: 87.0001 - val_r_square: -0.3219
Epoch 4/30
8/8 [==============================] - 0s 9ms/step - loss: 45.9980 - r_square: 0.1439 - val
_loss: 38.6999 - val_r_square: 0.4087
Epoch 5/30
8/8 [==============================] - 0s 10ms/step - loss: 29.3082 - r_square: 0.5080 - va
l_loss: 31.8937 - val_r_square: 0.5125
Epoch 6/30
8/8 [==============================] - 0s 11ms/step - loss: 20.8103 - r_square: 0.6347 - va
l_loss: 28.0669 - val_r_square: 0.5691
Epoch 7/30
8/8 [==============================] - 0s 9ms/step - loss: 16.2777 - r_square: 0.7205 - val
_loss: 22.9176 - val_r_square: 0.6545
Epoch 8/30
8/8 [==============================] - 0s 22ms/step - loss: 14.7423 - r_square: 0.7388 - va
l_loss: 17.6695 - val_r_square: 0.7329
Epoch 9/30
8/8 [==============================] - 0s 46ms/step - loss: 12.2654 - r_square: 0.7862 - va
l_loss: 15.3023 - val_r_square: 0.7665
Epoch 10/30
8/8 [==============================] - 0s 43ms/step - loss: 11.1046 - r_square: 0.8098 - va
l_loss: 14.8073 - val_r_square: 0.7732
Epoch 11/30
8/8 [==============================] - 0s 28ms/step - loss: 10.4192 - r_square: 0.8266 - va
l_loss: 12.8749 - val_r_square: 0.8019
Epoch 12/30
8/8 [==============================] - 0s 20ms/step - loss: 9.6957 - r_square: 0.8201 - val
_loss: 12.2551 - val_r_square: 0.8118
Epoch 13/30
8/8 [==============================] - 0s 11ms/step - loss: 9.5409 - r_square: 0.8222 - val
_loss: 12.4916 - val_r_square: 0.8085
Epoch 14/30
8/8 [==============================] - 0s 11ms/step - loss: 9.0431 - r_square: 0.8421 - val
_loss: 11.9993 - val_r_square: 0.8154
Epoch 15/30
8/8 [==============================] - 0s 11ms/step - loss: 8.8902 - r_square: 0.8404 - val
_loss: 12.1750 - val_r_square: 0.8131
Epoch 16/30
8/8 [==============================] - 0s 8ms/step - loss: 8.4721 - r_square: 0.8534 - val
```

```
loss: 11.9209 - val_r_square: 0.8167
Epoch 17/30
8/8 [==============================] - 0s 10ms/step - loss: 8.1633 - r_square: 0.8586 - val
_loss: 12.3309 - val_r_square: 0.8100
Epoch 18/30
8/8 [==============================] - 0s 11ms/step - loss: 8.1730 - r_square: 0.8585 - val
_loss: 11.9846 - val_r_square: 0.8158
Epoch 19/30
8/8 [==============================] - 0s 10ms/step - loss: 7.9933 - r_square: 0.8656 - val
_loss: 12.1346 - val_r_square: 0.8135
Epoch 20/30
8/8 [==============================] - 0s 9ms/step - loss: 7.9839 - r_square: 0.8478 - val_
loss: 13.2520 - val_r_square: 0.7964
Epoch 21/30
8/8 [==============================] - 0s 13ms/step - loss: 7.8978 - r_square: 0.8667 - val
_loss: 12.1168 - val_r_square: 0.8130
Epoch 22/30
8/8 [==============================] - 0s 11ms/step - loss: 8.1868 - r_square: 0.8672 - val
_loss: 12.6419 - val_r_square: 0.8062
Epoch 23/30
8/8 [==============================] - 0s 9ms/step - loss: 7.8425 - r_square: 0.8679 - val_
loss: 12.0707 - val_r_square: 0.8140
Epoch 24/30
8/8 [==============================] - 0s 13ms/step - loss: 7.7577 - r_square: 0.8680 - val
_loss: 12.7586 - val_r_square: 0.8043
Epoch 25/30
8/8 [==============================] - 0s 10ms/step - loss: 7.8928 - r_square: 0.8640 - val
_loss: 12.1913 - val_r_square: 0.8119
Epoch 26/30
8/8 [==============================] - 0s 10ms/step - loss: 7.7413 - r_square: 0.8684 - val
_loss: 12.2732 - val_r_square: 0.8113
Epoch 27/30
8/8 [==============================] - 0s 11ms/step - loss: 7.8519 - r_square: 0.8631 - val
_loss: 13.1755 - val_r_square: 0.7976
Epoch 28/30
8/8 [==============================] - 0s 12ms/step - loss: 8.0940 - r_square: 0.8644 - val
_loss: 12.3771 - val_r_square: 0.8097
Epoch 29/30
8/8 [==============================] - 0s 11ms/step - loss: 7.8811 - r_square: 0.8698 - val
_loss: 12.1618 - val_r_square: 0.8124
Epoch 30/30
8/8 [==============================] - 0s 11ms/step - loss: 7.9093 - r_square: 0.8634 - val
_loss: 13.1993 - val_r_square: 0.7965
```

In [ ]:

```
small_reg_network.evaluate(X1_test_norm, y1_test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 7.4710 - r_square: 0.8657
```

Out[ ]:

```
[7.470969200134277, 0.8656573295593262]
```

In [ ]:

```
#evaluate model
estimators = [('standardize', StandardScaler()),
              ('mlp', KerasRegressor(model = small_reg_model, epochs=30, verbose = 0))]
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X = X1, y = y1, cv=kfold, scoring='neg_mean_squared_er
ror')
print("Small: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Small: -8.93 (3.28) MSE
```

Observation:

- **We can observe above in our small network that it brought a small but noticeable lift to the MSE and r^2 of our**

optimized model. By adjusting the number of hidden layer node and making it 4, which is smaller than the input node, it reduced the MSE and increased r^2.**
- **We can compare the MSE of the kfold cross validation of this small network model to the previous model, the MSE of the previous model is -9.42 while the MSE of this small model is -8.93.**
- **This shows that having smaller network architecture decreases the MSE and increases r^2 of our model. This is a good result since having a smaller network means that we can train the model much faster than the previous model with 7 hidden layer nodes.**

- **For this larger network, I used 3 hidden layers which has 7, 4, and 2 hidden layer nodes respectively.**

In [ ]:

```
"""Larger network"""

def larger_reg_model():

  regmodel = tf.keras.models.Sequential([
      tf.keras.layers.Dense(7, input_shape=(7,), kernel_initializer = "normal",
                            activation = "relu"),
      tf.keras.layers.Dense(4, kernel_initializer = "normal", activation = "relu"),
      tf.keras.layers.Dense(2, kernel_initializer = "normal", activation = "relu"),
      tf.keras.layers.Dense(1,kernel_initializer = "normal",activation = "linear")
  ])

  regmodel.compile(optimizer = Adam(learning_rate = 0.1),
                loss = "mean_squared_error",
                metrics=[r_square]
                )
  return regmodel

larger_reg_network = larger_reg_model()

larger_reg_model_history2 = larger_reg_network.fit(X1_train_norm, y1_train,
                              validation_data = (X1_val_norm, y1_val),
                              epochs = 30)
```

```
Epoch 1/30
8/8 [==============================] - 3s 46ms/step - loss: 521.5256 - r_square: -8.2419 -
val_loss: 258.3245 - val_r_square: -2.9729
Epoch 2/30
8/8 [==============================] - 0s 23ms/step - loss: 109.3632 - r_square: -0.9327 -
val_loss: 89.5104 - val_r_square: -0.3770
Epoch 3/30
8/8 [==============================] - 0s 12ms/step - loss: 64.7959 - r_square: -0.2113 - v
al_loss: 64.8304 - val_r_square: 0.0076
Epoch 4/30
8/8 [==============================] - 0s 12ms/step - loss: 38.1060 - r_square: 0.3240 - va
l_loss: 35.6060 - val_r_square: 0.4545
Epoch 5/30
8/8 [==============================] - 0s 14ms/step - loss: 20.4213 - r_square: 0.6561 - va
l_loss: 23.7293 - val_r_square: 0.6386
Epoch 6/30
8/8 [==============================] - 0s 16ms/step - loss: 12.5152 - r_square: 0.7875 - va
l_loss: 19.6365 - val_r_square: 0.7021
Epoch 7/30
8/8 [==============================] - 0s 9ms/step - loss: 10.0868 - r_square: 0.8256 - val
_loss: 14.5890 - val_r_square: 0.7768
Epoch 8/30
8/8 [==============================] - 0s 9ms/step - loss: 10.3382 - r_square: 0.8173 - val
_loss: 12.7231 - val_r_square: 0.8050
Epoch 9/30
8/8 [==============================] - 0s 11ms/step - loss: 9.9835 - r_square: 0.8268 - val
_loss: 15.1684 - val_r_square: 0.7676
Epoch 10/30
8/8 [==============================] - 0s 12ms/step - loss: 9.5961 - r_square: 0.8350 - val
_loss: 18.8887 - val_r_square: 0.7121
Epoch 11/30
8/8 [==============================] - 0s 13ms/step - loss: 11.5609 - r_square: 0.7994 - va
l_loss: 14.2854 - val_r_square: 0.7824
```

```
Epoch 12/30
8/8 [==============================] - 0s 11ms/step - loss: 9.6144 - r_square: 0.8397 - val
_loss: 13.6255 - val_r_square: 0.7903
Epoch 13/30
8/8 [==============================] - 0s 17ms/step - loss: 8.4050 - r_square: 0.8522 - val
_loss: 13.6027 - val_r_square: 0.7915
Epoch 14/30
8/8 [==============================] - 0s 27ms/step - loss: 8.1950 - r_square: 0.8615 - val
_loss: 12.8256 - val_r_square: 0.8029
Epoch 15/30
8/8 [==============================] - 0s 23ms/step - loss: 8.1418 - r_square: 0.8620 - val
_loss: 13.5671 - val_r_square: 0.7919
Epoch 16/30
8/8 [==============================] - 0s 33ms/step - loss: 8.6818 - r_square: 0.8466 - val
_loss: 14.0018 - val_r_square: 0.7850
Epoch 17/30
8/8 [==============================] - 0s 26ms/step - loss: 7.9698 - r_square: 0.8554 - val
_loss: 13.8315 - val_r_square: 0.7879
Epoch 18/30
8/8 [==============================] - 0s 11ms/step - loss: 8.1667 - r_square: 0.8708 - val
_loss: 12.9846 - val_r_square: 0.8010
Epoch 19/30
8/8 [==============================] - 0s 6ms/step - loss: 7.8520 - r_square: 0.8572 - val_
loss: 13.7206 - val_r_square: 0.7895
Epoch 20/30
8/8 [==============================] - 0s 11ms/step - loss: 8.0357 - r_square: 0.8612 - val
_loss: 16.4267 - val_r_square: 0.7489
Epoch 21/30
8/8 [==============================] - 0s 6ms/step - loss: 8.7446 - r_square: 0.8507 - val_
loss: 17.6781 - val_r_square: 0.7300
Epoch 22/30
8/8 [==============================] - 0s 8ms/step - loss: 8.8436 - r_square: 0.8464 - val_
loss: 13.4357 - val_r_square: 0.7933
Epoch 23/30
8/8 [==============================] - 0s 6ms/step - loss: 8.4390 - r_square: 0.8512 - val_
loss: 15.6097 - val_r_square: 0.7579
Epoch 24/30
8/8 [==============================] - 0s 6ms/step - loss: 10.4284 - r_square: 0.8067 - val
_loss: 15.0949 - val_r_square: 0.7668
Epoch 25/30
8/8 [==============================] - 0s 8ms/step - loss: 10.2543 - r_square: 0.8264 - val
_loss: 13.5056 - val_r_square: 0.7933
Epoch 26/30
8/8 [==============================] - 0s 8ms/step - loss: 9.9035 - r_square: 0.8311 - val_
loss: 15.1931 - val_r_square: 0.7644
Epoch 27/30
8/8 [==============================] - 0s 8ms/step - loss: 7.8727 - r_square: 0.8662 - val_
loss: 13.0504 - val_r_square: 0.7992
Epoch 28/30
8/8 [==============================] - 0s 6ms/step - loss: 8.1056 - r_square: 0.8539 - val_
loss: 12.9335 - val_r_square: 0.8019
Epoch 29/30
8/8 [==============================] - 0s 8ms/step - loss: 7.9258 - r_square: 0.8663 - val_
loss: 13.7610 - val_r_square: 0.7869
Epoch 30/30
8/8 [==============================] - 0s 8ms/step - loss: 8.2893 - r_square: 0.8542 - val_
loss: 13.2291 - val_r_square: 0.7971
```

In [ ]:

```
larger_reg_network.evaluate(X1_test_norm, y1_test)
```

```
3/3 [==============================] - 0s 10ms/step - loss: 7.4977 - r_square: 0.8646
```

Out[ ]:

```
[7.497668266296387, 0.8645698428153992]
```

In [ ]:

```
#evaluate model
estimators = [('standardize', StandardScaler()),
             ('mlp', KerasRegressor(model = larger_reg_model, epochs=30, verbose = 0))]
```

```
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X = X1, y = y1, cv=kfold, scoring='neg_mean_squared_er
ror')
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Larger: -32.44 (32.70) MSE
```

Observation:

- **Here we can see that this larger network performed much worse than the optimized and smaller network. This indicates that having more hidden layer and hidden layer nodes does not mean that it will result to having a well-fitted model.**
- **We can also observe in the results of the training of the model that the MSE and r^2 fluctuates frequently, which means that having more layers and more hidden nodes may result to more complex but not accurate model.**
- **The result of the kfold cross validation for the larger network is -32.44 with an std of 32.70. This is much worse than the result of the kfold cross validation of the optimized model and the smaller network model. The MSE of the larger network is 3 times the MSE of the smaller network model.**
- **Overall, having a larger network architecture is not suitable for this dataset. We can also see the number of hidden layers and hidden layer nodes is not directly proportional to the performance of the model. This model takes much longer to train but it has worse results than the 2 previous models.**

SUMMARY AND CONCLUSION:

Summary:

- **For this activitiy, we were tasked to perform different steps of tuning and evaluation for classification and regression dataset.**

1. **Classification Task**

   - **For the classification task, I picked the Breast Cancer Wisconsin dataset and performed EDA, in which I explored the data and found out that there are no missing values and I also found out the correlation of each variable to the target variable. The next thing I did is to pre-process the data, I removed the ID column since it is just a unique identifier and does not contribute to the prediction of the target class, I also used lambda function to convert the categorical target variable "Diagnosis" to 0 and 1 respectively. The positive class for this dataset or the class that we're testing for is the 0 or the malignant, while the negative class for this dataset is 1 which is the benign. I split the dataset into training, validation and test which has a split of 60/20/20. The next thing I did is to normalize the dataset using the standard scaler.**
   - **In training the model for classification task, I only used 1 hidden layer which is made up of 4 hidden layer nodes which is 2/3 of the 7 input layer nodes. I used relu activation function and normal for the kernel initializer. For the compilation of the model, I used adam as the optimized with a learning rate of 0.001, I used binary crossentropy for the loss function and accuracy for the metric. I only used 30 epochs based on arbitrary decision which can be adjusted based on the results.**
   - **The baseline model for the classification dataset performed very well with an accuracy of 0.94 and loss of 0.1. I used kfold cross validation to further test the model, it resulted to 0.969 accuracy, This shows that the model performs very well and there is a slight overfitting because the training model has 0.99 accuracy.**
   - **To further optimize the baseline model for classificaion, I did some fine tuning of the hidden layers and added dropout layers to further reduce overfitting. The number of epoch stays the same and the learning rate also stays the same. After testing the model, the accuracy increased to 0.96 and loss of 0.1. The kfold cross validation scores is 0.976 which indicates that this model performed better than the previous model.**
   - **Overall, I can say that the final model is a well fitted because the accuracy of the testing and the training only has a small gap. The accuracy is also within the standard of 0.95.**

2. **Regression Task**

- For the regression task, I picked the Auto MPG dataset which is about the Miles per Gallon of cars. The goal of this dataset is to predict the MPG based on the different features. The first thing I did is to perform EDA and found out that there are missing values for the horsepower column, so I filled it the missing values with the mean of the horsepower column. The next thing I did is to delete the "car_names" column which is just a unique identifier for the cars. Then I visualized the correlation using heatmap, and found out that there are variables which has moderate correlation with the target variable MPG. Then I split the dataset into training, validation and testing sets in 60/20/20 ratio. Then I normalized the dataset using standard scaler.
- For the baseline model of regression, I used the original dataset and not the standardized dataset because I will be optimizing this model using the standardized dataset later. I used 1 hidden layer and 7 hidden layer nodes for this model, the hidden layer node is equal to the number of the input layer nodes. The activation for the hidden layer is relu but for the output layer it is linear activation function, this is because the task is for regression where the values of the output are continuous.
- The result of the training of baseline model is a high loss and negative r^2 which indicates a horrible model. I used kfold cross validation to further evaluate the model, and it still showed a high MSE of -62 which is a bad score since it is not close to 0. To visualize the model, I plotted the linear regression and found out that there is no regresion line for this model and just by looking at the plot, we can see that it does not give us any significant information.
- To optimize the baseline model, I normalized the data using standard scaler and I used that for the training and testing of this model. I did not change anything with the configuration of the model and the epoch that I used is still 30 and the learning rate is still 0.1 The result of the optimized model is astonishing because the loss and r^2 significantly improved. The loss or the MSE is now only 7 compared to the MSE of the baseline model which is 80, and the r^2 is now 0.8 compared to the -0.5 r^2 of the baseline model. The closer the r^2 to 1, the better the model. I also used kfold cross validation and the score is -9.42 negative mse. This is a good score compared to the previous -62 MSE since -9.42 is significantly closer to 0. For the plotted linear regression, we can see that there is now a line and the predicted and observed data points are clustered in around that line. This shows a good regression model since this indicates that our model successfully predicted the values of the testing set.
- In fine tuning of the hidden layers, I used the small network and large network architecture. In the small network architecture, I used 1 hidden layer with a hidden layer node of 4, which is half the input layer node. Other than that, I did not change anything and the rest of the parameters stayed the same. After training small network architecture, the MSE and the r^2 slight improved compared to the optimized model. The kfold cross validation score of the small network architecture is -8.93 which is closer to 0 than the previous model. This is a good result since we can train the model in half the time of the previous optimized model and get an even better result.
- For the larger network architecture, I used 3 hidden layers with 7,4,2 hidden layer nodes respectively. The rest of the parameters stayed the same. After training the model, I found out that it has bad results compared to the smaller network architecture. For the k fold cross validation, the score is -32.44 which shows that the negative mse worsened because the value moved further away from 0. This shows that the number of hidden layer and hidden layer nodes is not directly proportional to the performance of the model.
- Overall, in this task, the optimized model and the smaller network model performed the best. The smaller network model architecture has a slight advantage to the optimized model since it uses half the time to be trained compared to the optimized model. The optimized model and the smaller network model are well fitted with this dataset.

Conclusion:

- In conclusion, this activity gave me a lot of insight regarding the difference between creating models that are suited for classification and regresion task. This activity also challenged my knowledge regarding Exploratory Data Analysis and the difference in performing EDA in classification and regression task. The different optimization and fine tuning that I did also enlightened me regarding the methods that I can use to improve different models that I will be coding in the future. This activity also gave me new methods in evaluating the models like the k-fold cross evaluation and the smaller/larger network architecture. Overall, I enjoyed this activity as it trained me to become a better data scientist and it also gave me a lot of new information that can be useful when applied to different datasets and models. I will certainly recommend this activity to future data science students because this can build their foundation in data engineering.

REFERENCES

- [1]I. Guyon, “A scaling law for the validation-set training-set size ratio.” Accessed: Nov. 05, 2023. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=452e6c05d46e061290fefff8b46d0ff161998677
- [2]“Splitting into train, dev and test sets,” cs230.stanford.edu. https://cs230.stanford.edu/blog/split/
- [3]I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016
- [4]S. Karsoliya, “Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture,” 2012. Accessed: Mar. 24, 2024. [Online]. Available: https://ijettjournal.org/volume-3/issue-6/IJETT-V3I6P206.pdf
- [5]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, and Y. Bengio, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014, Available: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Link for datasets

- Classification Dataset
  - Wolberg,William, Mangasarian,Olvi, Street,Nick, and Street,W.. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.
- Regression Dataset
  - Quinlan,R.. (1993). Auto MPG. UCI Machine Learning Repository. https://doi.org/10.24432/C5859H.