Homework 6

Before you start: Read Chapter 10 Logistic Regression and Chapter 11 Neural Networks in the textbook.

Note: Please enter the code along with your comments in the **TODO** section.

Alternative solutions are always welcomed.

```
# # Please remove # and run the following code if you have an error while importing the dataset
!pip install --upgrade openpyxl
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: openpyxl in /usr/local/lib/python3.9/dist-packages (3.0.10)
Collecting openpyxl
  Downloading openpyxl-3.1.2-py2.py3-none-any.whl (249 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 250.0/250.0 kB 9.3 MB/s eta 0:00:00
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.9/dist-packages (from openpyxl) (1.1.0)
Installing collected packages: openpyxl
  Attempting uninstall: openpyxl
    Found existing installation: openpyxl 3.0.10
    Uninstalling openpyxl-3.0.10:
      Successfully uninstalled openpyxl-3.0.10
Successfully installed openpyxl-3.1.2
```

Part 1: Logistic Regression

Problem 3 - Financial Condition of Banks

The file **Banks.csv** includes data on a sample of 20 banks.

The "Financial Condition" column records the judgment of an expert on the financial condition of each bank. This response variable takes one of two possible values—weak or strong—according to the financial condition of the bank.

The predictors are two ratios used in the financial analysis of banks: TotLns&Lses/Assets is the ratio of total loans and leases to total assets and TotExp/Assets is the ratio of total expenses to total assets.

The target is to classify the financial condition of a new bank using the two ratios.

Note: 1 for financially weak banks and 0 otherwise

```
# Load the required packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import mean_squared_error
```

```
# Import the dataset
#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

Choose Files 4 files

- **Banks.csv**(text/csv) - 461 bytes, last modified: 3/27/2019 - 100% done
- **insurance.csv**(text/csv) - 55628 bytes, last modified: 3/30/2023 - 100% done
- **Problem1.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 8990 bytes, last modified: 3/21/2023 - 100% done
- **SystemAdministrators.csv**(text/csv) - 797 bytes, last modified: 3/27/2019 - 100% done

```
Saving Banks.csv to Banks.csv
```

TODO 1

Run a logistic regression model (on the entire dataset) that models the status of a bank as a function of the two financial measures provided.

```
df = pd.read_csv("Banks.csv")
df.head()
```

	Obs	Financial Condition	TotCap/Assets	TotExp/Assets	TotLns&Lses/Assets
0	1	1	9.7	0.12	0.65
1	2	1	1.0	0.11	0.62
2	3	1	6.9	0.09	1.02
3	4	1	5.8	0.10	0.67
4	5	1	4.3	0.11	0.69

```
X = df.drop(["Financial Condition", "TotCap/Assets", "Obs"], axis = 1)
y = df["Financial Condition"]
X.shape, y.shape
```

```
((20, 2), (20,))
```

```
clf = LogisticRegression().fit(X, y)
clf.score(X, y)
```

```
0.75
```

TODO 2

Write the estimated equation that associates the financial condition of a bank with its two predictors in three formats:

a. The logit as a function of the predictors

b. The odds as a function of the predictors

c. The probability as a function of the predictors

```
print("Log Odds: ",clf.coef_[0])
print("Odds: ",np.exp(clf.coef_[0]))
print("Intercept: ", clf.intercept_[0])
```

```
Log Odds:  [0.16075579 0.72642506]
Odds:  [1.17439814 2.06767557]
Intercept:  -0.4733365221513398
```

```
def logit(y_test, coef, intercept):
  return np.dot(y_test, coef) + intercept
def odds(y_test, coef, intercept):
  return np.exp(np.dot(y_test, coef) + intercept)
def prob(y_test, coef, intercept):
  return np.exp(np.dot(y_test, coef) + intercept)/(1 + np.exp(np.dot(y_test, coef) + intercept))
```

a. Logit = -0.4733365221513398 + 0.16075579 x TotExp/Assets + 0.72642506 x TotLns&Lses/Assets

b. Odds = exp(-0.4733365221513398 + 0.16075579 x TotExp/Assets + 0.72642506 x TotLns&Lses/Assets)

c. p = 1/(1 + exp(0.4733365221513398 - 0.16075579 x TotExp/Assets - 0.72642506 x TotLns&Lses/Assets))

TODO 3

Consider a new bank whose total loans and leases/assets ratio = 0.6 and total expenses/assets ratio = 0.11.

From your logistic regression model, estimate the following four quantities for this bank:

the logit, the odds, the probability of being financially weak, and the classification of the bank (use cutoff = 0.5).

```
y_test = np.array([0.11, 0.6])
print("Logit: ", logit(y_test,clf.coef_[0], clf.intercept_[0]))
print("Odds: ", odds(y_test,clf.coef_[0], clf.intercept_[0]))
print("probability of being financially weak: ", prob(y_test,clf.coef_[0], clf.intercept_[0]))
```

```
Logit:  -0.019798345919363358
Odds:  0.9803963542999818
probability of being financially weak:  0.4950505751897964
```

As below cutoff value of 0.5 it is financly strong

TODO 4

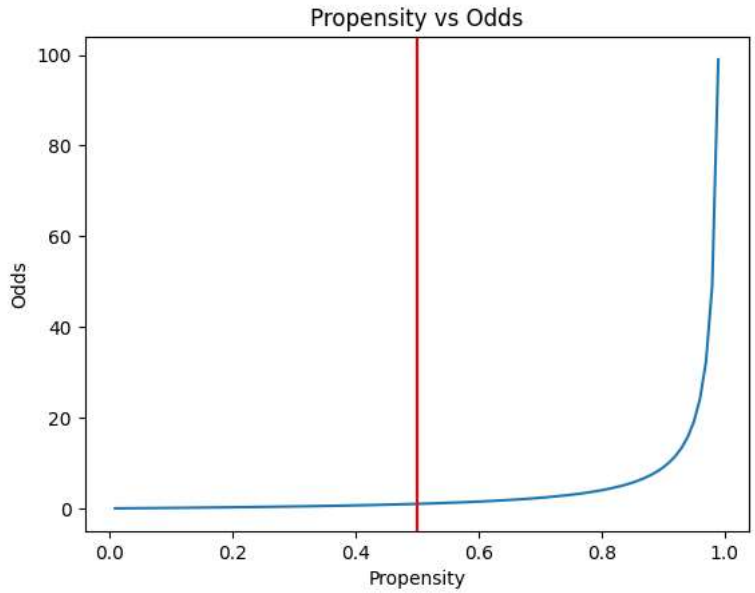
We use a cutoff value of 0.5 to classify a record based on propensity.

Instead, if we want to classify the record using the odds or logit, what value should we take as a cutoff?

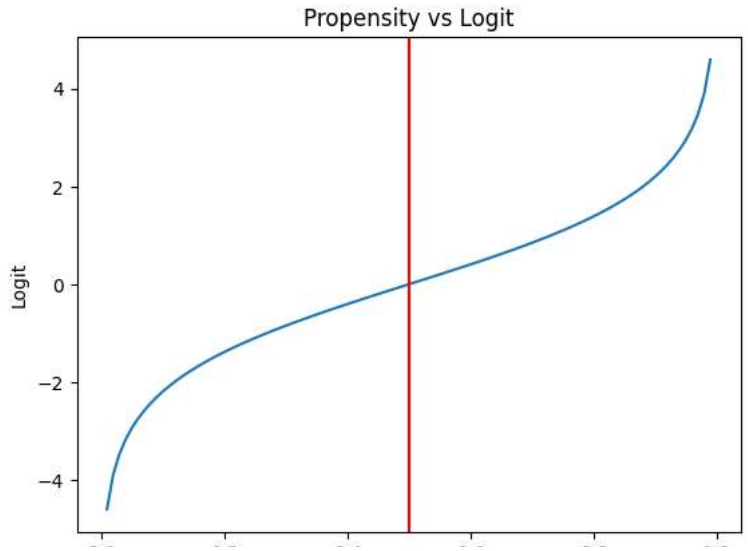
```
def plot_fig(x,y, y_label):
  graph = sns.lineplot(x=x, y=y)
  graph.axvline(0.5, color='r')
  plt.xlabel("Propensity")
  plt.ylabel(y_label)
  plt.title("Propensity vs " + y_label)
  plt.show()
```

```
propensity = []
odds = []
logit = []
for p in np.arange(0.01,1,0.01):
  propensity.append(p)
  odds.append(p/(1-p))
  logit.append(np.log(p/(1-p)))
```

```
plot_fig(propensity, odds, y_label = "Odds")
```




```
plot_fig(propensity, logit, y_label = "Logit")
```



- If we go by odds from graph we can observe that morethan 1 value as class of interest and less than 1 as alternate class
- Similarly from graph positive value as class of interest and less than 0 as alternate class for logit

TODO 5

When a bank with in poor financial condition is misclassified as financially strong, the misclassification cost is much higher than a financially strong bank misclassified as weak.

To minimize the expected cost of misclassification, should the cutoff value for classification (which is currently at 0.5) be increased or decreased?

- By decreasing the probability threshold or cutoff value for classification, we can classify more banks as financially weak, which reduces the chances of misclassifying a weak bank as strong. However, this will increase the chances of misclassifying a strong bank as weak.

Problem 4 - Identifying Good System Administrators

A management consultant is studying the roles played by experience and training in a system administrator's ability to complete a set of tasks in a specified amount of time. In particular, the consultant is interested in discriminating between administrators who are able to complete given tasks within a specified time and those who are not.

Data are collected on the performance of 75 randomly selected administrators. They are stored in the file **SystemAdministrators.csv**.

The variable Experience measures months of full-time system administrator experience, while Training measures the number of relevant training credits. The outcome variable Completed is either Yes or No, according to whether or not the administrator completed the tasks.

```
# Import the dataset
df1 = pd.read_csv("SystemAdministrators.csv")
df1
```

	Experience	Training	Completed task
0	10.9	4	Yes

```
df1["Completed task"].value_counts()
```

```
No     60
Yes    15
Name: Completed task, dtype: int64
```

4	9.4	8	Yes

TODO 1

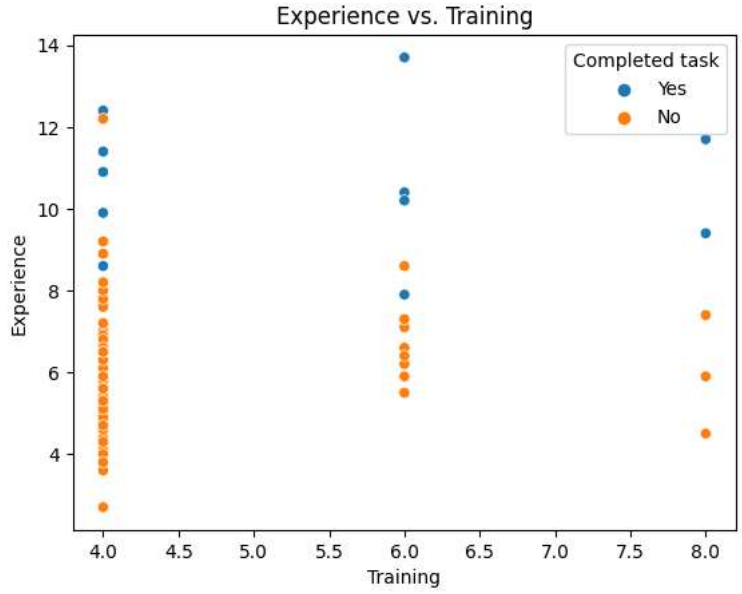
Create a scatter plot of Experience vs. Training using color or symbol to distinguish the administrators' task completion statues.

Which predictor(s) appear(s) potentially useful for the classifying task?

72	6.4	6	No

```
sns.scatterplot(data=df1, x="Training", y="Experience", hue="Completed task")
plt.title("Experience vs. Training")
```

```
Text(0.5, 1.0, 'Experience vs. Training')
```



Double-click (or enter) to edit

From the graph we can see that if we draw a horizontal line at around 8 years of experience we can almost split the status of completion of task.

So i would pick Experience as a predictor

TODO 2

Run a logistic regression model with both predictors using the entire dataset as training data. Among those who completed the task, what is the percentage of administrators incorrectly classified as failing to complete the task?

```
X = df1.drop(["Completed task"], axis = 1)
y = df1["Completed task"]
X.shape, y.shape
```

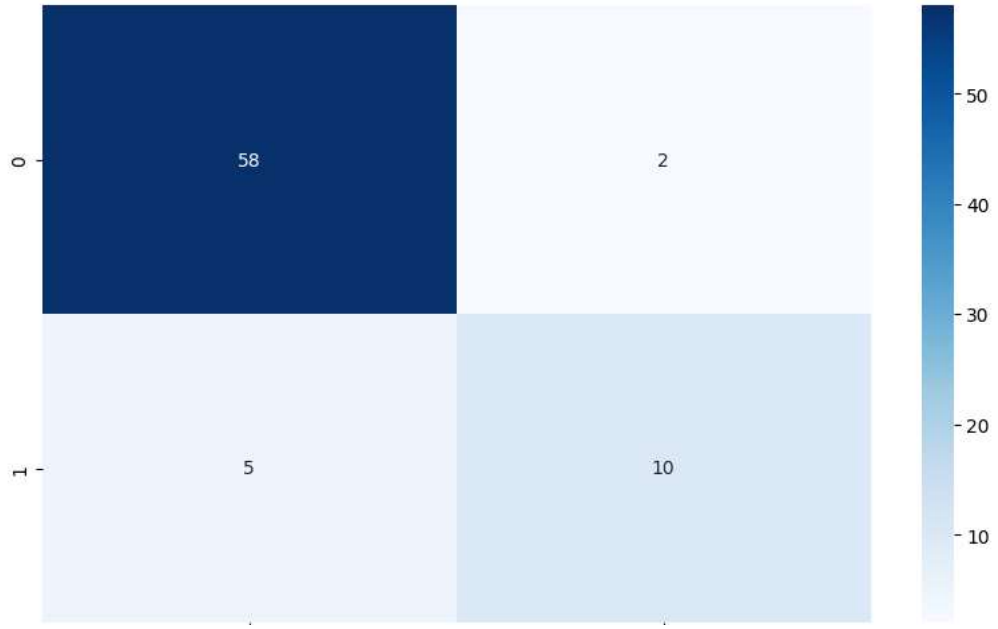
```
((75, 2), (75,))
```

```
clf1 = LogisticRegression().fit(X, y)
y_pred = clf1.predict(X)
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
print(classification_report(y, y_pred), '\n')
```

```
              precision    recall  f1-score   support

          No       0.92      0.97      0.94        60
         Yes       0.83      0.67      0.74        15

    accuracy                           0.91        75
   macro avg       0.88      0.82      0.84        75
weighted avg       0.90      0.91      0.90        75
```



From the above image

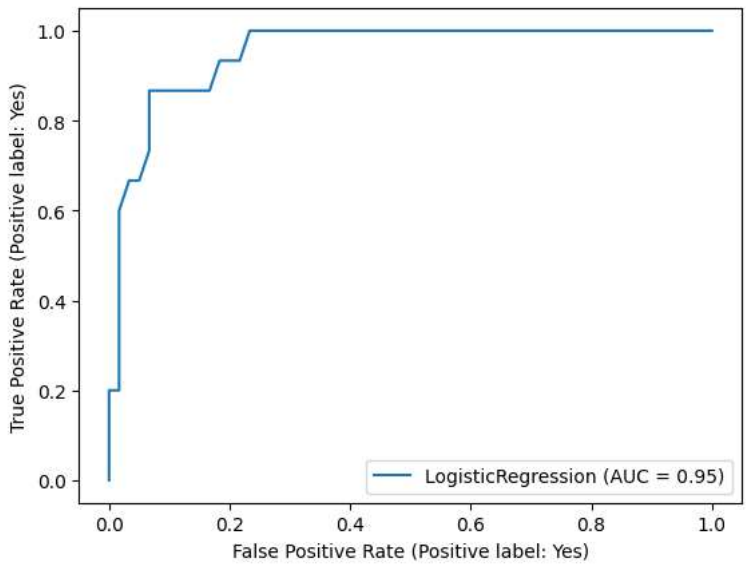
- Among those who completed the task, what is the percentage of administrators incorrectly classified as failing to complete the task?
- Percentage = 5/15 => 33.33%

TODO 3

To decrease the percentage in TODO 2, should we increase or decrease the cutoff probability?

```
RocCurveDisplay.from_estimator(estimator = clf1, X = X, y = y)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f84a6d0a370>
```



The percentage we calculated is FPR and along x-axis Cutoff decreases when FPR goes from 0 to 1.

- So to decrease the percentage 0.33 we are coming towards zero which implies cutoff increases towards 1.
- => Increase cutoff

TODO 4

How much experience must be accumulated by a administrator with 4 years of training before his or her estimated probability of completing the task exceeds 0.5?

```
clf1.coef_[0], clf1.intercept_[0]
```

```
(array([1.04592814, 0.1640162 ]), -10.245229250673233)
```

The activation function in logistic function is sigmoid function and for the probability to cross more than 0.5 implies the Z > 0

- => -10.245229250673233 + 1.04592814 x Experience + 0.1640162 x Training > 0
- Implies Experience > 9.168091080017 months

Part 2: Decision Trees

Problem 5 - Wine Data

The wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. (For illustration simplicity purpose, only 2 classes, 0 and 1, will be included for the classification task.) The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The objective is to classify the wines into class 0 or 1 using the 13 given attributes and a decision tree classifier.

```
from sklearn import datasets
import pandas as pd

# load the wine dataset
wine = datasets.load_wine()
print(wine.DESCR)

# convert the data into dataframe format
X = pd.DataFrame(wine['data'], columns = wine['feature_names'])
y = wine['target']

# only consider wine class 0 and 1
X = X.loc[0:129, :]
y = y[0:130]

X.head()
```

```
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================== ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================== ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================== ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyan
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2

TODO 1

Partition the data into 70% training and 30% validation set.

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((91, 13), (39, 13), (91,), (39,))
```

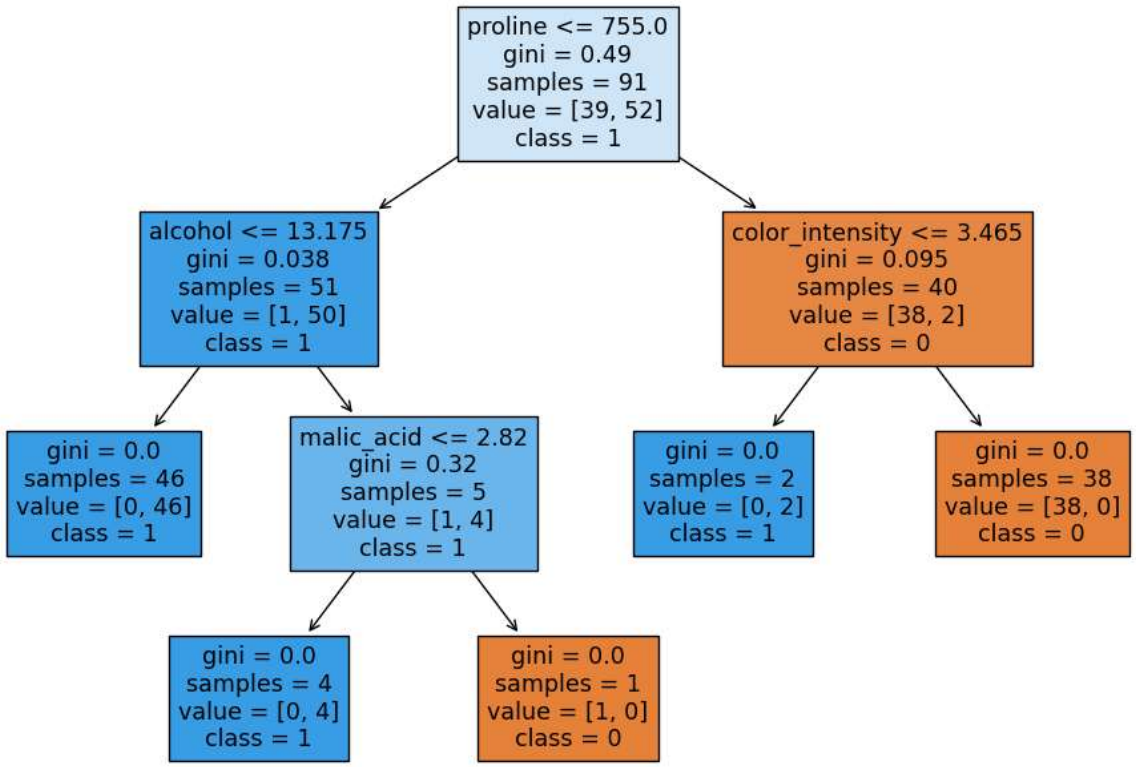
TODO 2

Fit a decision tree classifier on the training set with no pruning.

Plot the tree with the following requirements:

- The node with splitting rule should contain variable name instead of variable index.
- Pick the appropriate information to present in the node. The node should be of appropriate size so the information is clear for viewing.
- The node should be colored.

```
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)
plt.figure(figsize=(12,8))
fig = tree.plot_tree(clf2, feature_names= X_train.columns, class_names = np.unique(y_train).astype(str), filled=True)
```



Hint: Decision tree classifier with sklearn

TODO 3

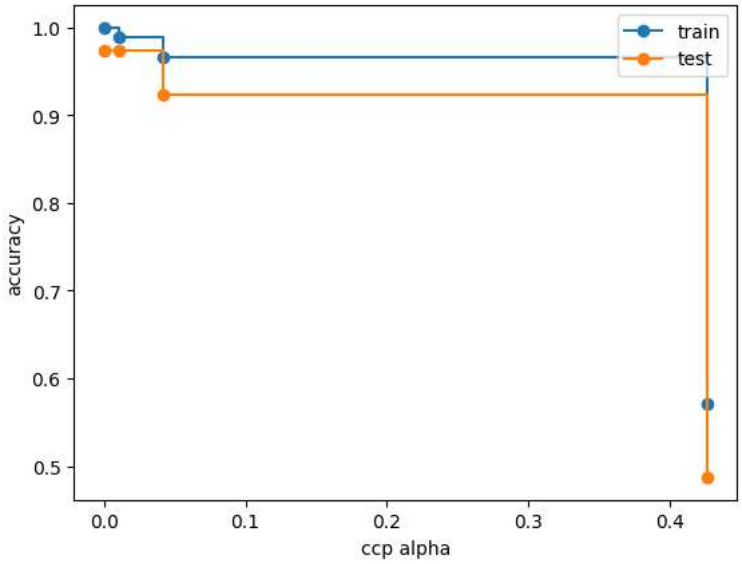
Prune the tree with cost complexity. What is the best ccp value? Use visualization to back up your decision.

Plot the pruned tree in the same manner as TODO 2.

```
path = clf2.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Fit decision trees with different ccp values
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
```

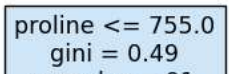
```
# Determine accuracy scores for different ccp values
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
# Plot accuracy scores vs ccp values
plt.plot(ccp_alphas, train_scores, marker='o', label='train', drawstyle="steps-post")
plt.plot(ccp_alphas, test_scores, marker='o', label='test', drawstyle="steps-post")
plt.xlabel("ccp alpha")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```




```
# Select best ccp value
best_ccp_alpha = ccp_alphas[test_scores.index(max(test_scores))]
print("best_ccp_alpha =",best_ccp_alpha,"\n")
# Fit decision tree with best ccp value
pruned_tree = DecisionTreeClassifier(random_state=42, ccp_alpha=best_ccp_alpha)
pruned_tree.fit(X_train, y_train)

# Plot pruned tree
plt.figure(figsize=(12,8))
fig = tree.plot_tree(pruned_tree, feature_names= X_train.columns, class_names = np.unique(y_train).astype(str), filled=True)
plt.show()
```

```
best_ccp_alpha = 0.0
```



Hint: Minimal cost complexity pruning

Post pruning decision trees with cost complexity with sklearn

Part 3: Neural Network

Problem 6 - Insurance amount prediction

Consider the insurance data (insurance.csv). The dataset consists of age, sex, BMI(body mass index), children, smoker and region feature, which are independent and "**charge**" as a target feature. We will predict individual medical costs billed by health insurance.

```
!pip install tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.9/dist-packages (2.12.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.2.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.4.8)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.53.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.1)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow) (67.6.1)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/pythor
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.32.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (16.0.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.9/dist-packages (from astunparse>=1.6.0->tensorflow) (0.40.0
Requirement already satisfied: ml-dtypes>=0.0.3 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow) (0.0.4)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow) (1.10.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow) (3.4
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.2
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->te
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->t
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard<
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from google-auth-oauthlib<1.1,>=0.5->
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.9/dist-packages (from markdown>=2.6.8->tensorboard<2.13
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.1
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensc
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oau
```

```
# Import the dataset
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential
```

```
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
```

Todo 1:

1. Load the insurance csv into a dataframe.
2. preprocess and clean the data by checking for null values.
3. Concert all the categorical variables to numerical and perform one-hot encoding.

```
# Import the dataset
df2 = pd.read_csv("insurance.csv")
df2
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

```
df2.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

```
enc = OneHotEncoder(sparse=False)
columns_to_one_hot = ['sex','smoker','region']
encoded_array = enc.fit_transform(df2.loc[:,columns_to_one_hot])
df_encoded = pd.DataFrame(encoded_array,columns=enc.get_feature_names_out() )
df_sklearn_encoded = pd.concat([df2,df_encoded],axis=1)
df_sklearn_encoded.drop(labels= columns_to_one_hot,axis=1,inplace=True)
df_sklearn_encoded
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `spar
  warnings.warn(
```

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	re
0	19	27.900	0	16884.92400	1.0	0.0	0.0	1.0	0.0	0.0	

Todo 2:

Separate the X and y for predictors and target variable and create a train test split of ratio 80, 20.

Note: TARGET_NAME = 'charges'

```
X = df_sklearn_encoded.drop(["charges"],axis=1)
y = df_sklearn_encoded["charges"]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1070, 11), (268, 11), (1070,), (268,))
```

Todo 3:

1. build a neural network model with 2 layers.

   Layer 1: first layer should contian 10 nodes. You are free to choose any activation function for the first layer.

   Layer 2: Use your intuition to decide on the number of nodes and activation function that can be used for layer 2. (You can research about the output layer regression models for neural networks)

2. Train the model for 2000 iterations.

Let this be model1

Ref: https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/

```
# create scaler object to scale data
standard_scaler  = StandardScaler()

# scale the training data
X_train_scaled = standard_scaler.fit_transform(X_train)
X_test_scaled = standard_scaler.transform(X_test)

# Define model1
model1 = Sequential()
model1.add(Dense(10, input_dim=X_train.shape[1], activation='relu')) # Layer 1 with 10 nodes and ReLU activation
model1.add(Dense(1, activation='linear')) # Layer 2 with 1 node and linear activation (output layer)
# Compile model1
model1.compile(loss='mean_squared_error', optimizer='adam')
# Train model1
history1 = model1.fit(X_train, y_train, epochs=2000, batch_size=32, validation_split=0.2, verbose=1)
```

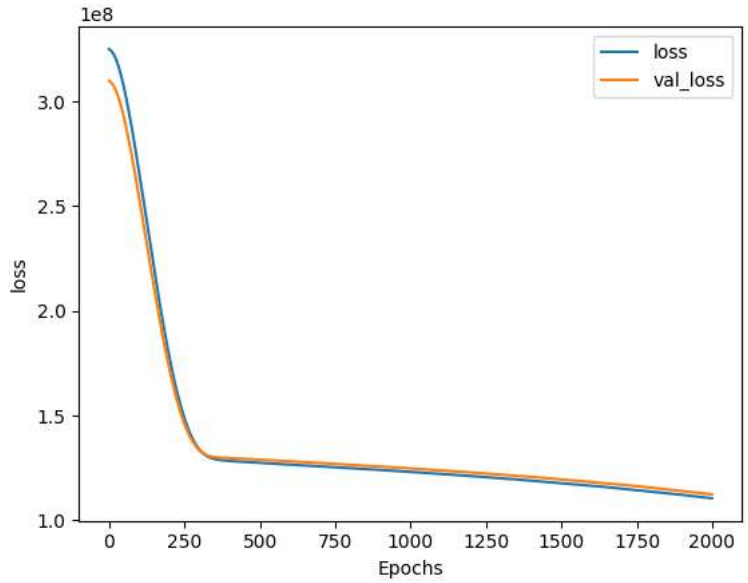
```
Epoch 1985/2000
27/27 [==============================] - 0s 7ms/step - loss: 110679424.0000 - val_loss: 112558528.0000
Epoch 1986/2000
27/27 [==============================] - 0s 6ms/step - loss: 110664896.0000 - val_loss: 112539304.0000
Epoch 1987/2000
27/27 [==============================] - 0s 8ms/step - loss: 110646224.0000 - val_loss: 112525016.0000
Epoch 1988/2000
27/27 [==============================] - 0s 8ms/step - loss: 110629312.0000 - val_loss: 112507112.0000
Epoch 1989/2000
27/27 [==============================] - 0s 9ms/step - loss: 110611632.0000 - val_loss: 112490320.0000
Epoch 1990/2000
27/27 [==============================] - 0s 8ms/step - loss: 110593648.0000 - val_loss: 112472792.0000
Epoch 1991/2000
27/27 [==============================] - 0s 8ms/step - loss: 110579856.0000 - val_loss: 112454328.0000
Epoch 1992/2000
27/27 [==============================] - 0s 7ms/step - loss: 110560240.0000 - val_loss: 112438128.0000
Epoch 1993/2000
27/27 [==============================] - 0s 6ms/step - loss: 110542560.0000 - val_loss: 112419744.0000
Epoch 1994/2000
27/27 [==============================] - 0s 6ms/step - loss: 110527256.0000 - val_loss: 112406056.0000
Epoch 1995/2000
27/27 [==============================] - 0s 5ms/step - loss: 110507344.0000 - val_loss: 112388648.0000
Epoch 1996/2000
27/27 [==============================] - 0s 7ms/step - loss: 110491384.0000 - val_loss: 112369624.0000
Epoch 1997/2000
27/27 [==============================] - 0s 6ms/step - loss: 110472680.0000 - val_loss: 112354320.0000
Epoch 1998/2000
27/27 [==============================] - 0s 6ms/step - loss: 110456672.0000 - val_loss: 112335224.0000
Epoch 1999/2000
27/27 [==============================] - 0s 6ms/step - loss: 110439280.0000 - val_loss: 112323304.0000
Epoch 2000/2000
27/27 [==============================] - 0s 5ms/step - loss: 110420536.0000 - val_loss: 112303152.0000
```

Todo 4:

Plot the training and validation loss for model1.

```
def plot_history(history_data, key):
  plt.plot(history_data.history[key])
  plt.plot(history_data.history['val_'+key])
  plt.xlabel("Epochs")
  plt.ylabel(key)
  plt.legend([key, 'val_'+key])
  plt.show()


# Plot the history
plot_history(history1, 'loss')
```



Todo 5:

build a deep neural network model with 4 layers.

Layer 1: first layer should contian 50 nodes. You are free to choose any activation function for the first layer.

Layer 2: first layer should contian 30 nodes. You are free to choose any activation function for the first layer.

Layer 3: first layer should contian 20 nodes. You are free to choose any activation function for the first layer.

Layer 4: Use your intuition to decide on the number of nodes and activation function that can be used for layer . (You can research about the output layer regression models for neural networks)

Train the model for 5000 iterations.

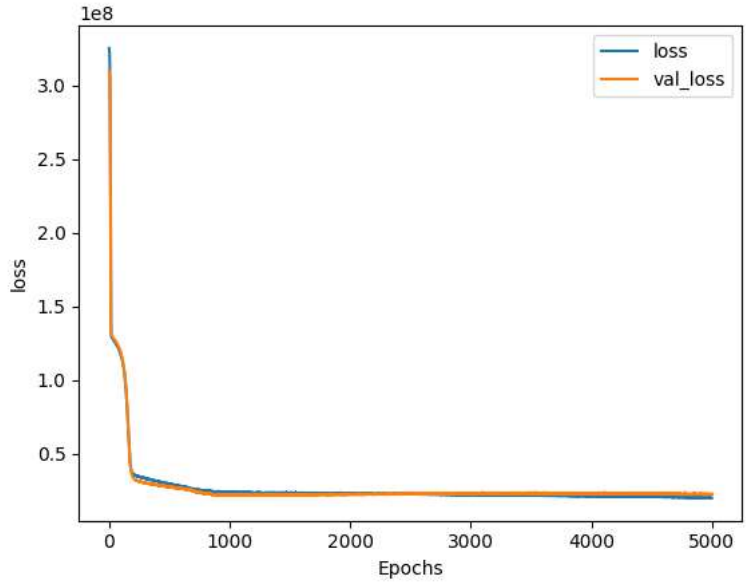
Let this be model2

```
# create model2
model2 = Sequential()
model2.add(Dense(50, input_dim=X_train.shape[1], activation='relu')) # layer1
model2.add(Dense(30, activation='relu')) # layer 2
model2.add(Dense(20, activation='relu')) # layer 3
model2.add(Dense(1)) # layer 4
# compile model2
model2.compile(loss='mean_squared_error', optimizer='adam')
# train model2
history2 = model2.fit(X_train, y_train, epochs=5000, batch_size=64, validation_split=0.2, verbose=0)
```

Todo 6:

Plot the training and validation loss for model2.

```
plot_history(history2, 'loss')
```



Todo 7: Comparing model 1 and 2.

1. predict the charges for test data using model 1 and 2.
2. using mse as performance evaluation metrics, comment of model performance for the test data.
3. Compare the performance of model 1 and 2.

What do you understand from the models built.

```
def predict_info(model, X_test, y_test):
  y_pred = model.predict(X_test)
  mse = mean_squared_error(y_test, y_pred)
  return y_pred, mse


# Predicting charges for test data using model 1
y_pred1, mse1 = predict_info(model1, X_test, y_test)
print("MSE for model 1:", mse1)
```

```
9/9 [==============================] - 0s 2ms/step
MSE for model 1: 114180829.59666839
```

```
# Predicting charges for test data using model 2
y_pred2, mse2 = predict_info(model2, X_test, y_test)
print("MSE for model 2:", mse2)
```

```
9/9 [==============================] - 0s 4ms/step
MSE for model 2: 20931023.799519982
```

```
mse1>mse2
```

```
True
```

- From the MSE we can say that model2 performance is good than model1
- It is obvious that as we increased no of iterations and layers the predictive ability of a neural network improves. But a con is it also increases training resourses consumed.

*Todo 8: *

convert the target variable into categorical by using median as cutoff.

Use the created variable as the new target variable and drop charges column from your predictors.

```
data_masked = df_sklearn_encoded.copy()
median = data_masked['charges'].median()
data_masked['charges'] = np.where(data_masked['charges'] <= median, 0, 1)
data_masked
```

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	regio
0	19	27.900	0	1	1.0	0.0	0.0	1.0	0.0	0.0	
1	18	33.770	1	0	0.0	1.0	1.0	0.0	0.0	0.0	
2	28	33.000	3	0	0.0	1.0	1.0	0.0	0.0	0.0	
3	33	22.705	0	1	0.0	1.0	1.0	0.0	0.0	1.0	
4	32	28.880	0	0	0.0	1.0	1.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	
1333	50	30.970	3	1	0.0	1.0	1.0	0.0	0.0	1.0	
1334	18	31.920	0	0	1.0	0.0	1.0	0.0	1.0	0.0	
1335	18	36.850	0	0	1.0	0.0	1.0	0.0	0.0	0.0	
1336	21	25.800	0	0	1.0	0.0	1.0	0.0	0.0	0.0	
1337	61	29.070	0	1	1.0	0.0	0.0	1.0	0.0	1.0	

1338 rows × 12 columns

Todo 9:

Seperate the X and y for predictors and target variable and create a train test split of ratio 80, 20.

```
X = data_masked.drop(["charges"],axis=1)
y = data_masked["charges"]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1070, 11), (268, 11), (1070,), (268,))
```

Todo 10:

Build a neural network model with 3 layers.

Layer 1: first layer should contian 20 nodes. You are free to choose any activation function for the first layer.

Layer 2: first layer should contian 10 nodes. You are free to choose any activation function for the first layer.

Layer 3: Use your intuition to decide on the number of nodes and activation function that can be used for layer 3. (You can research about the output layer classification models for neural networks)

Train the model for 2000 iterations. Use binary_crossentropy for loss function.

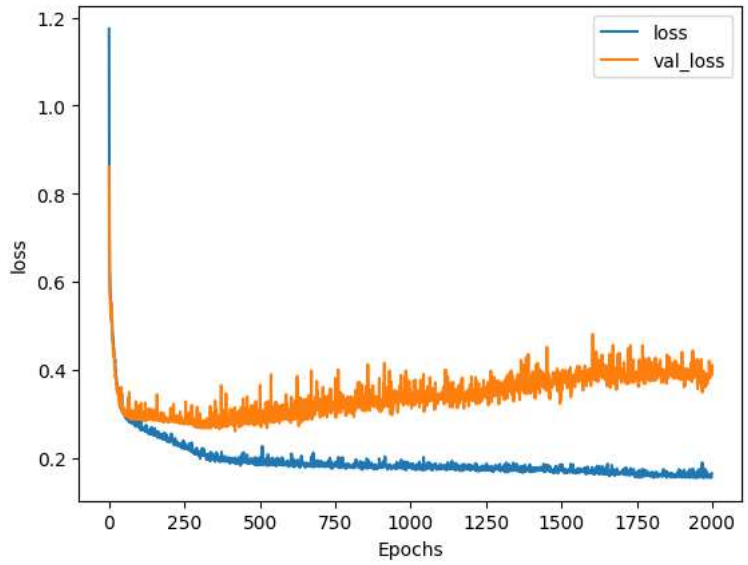
Let this be model3

Ref: https://www.analyticsvidhya.com/blog/2021/11/neural-network-for-classification-with-tensorflow/

```
# Define the model
model3 = Sequential()
model3.add(Dense(20, input_shape=(X_train.shape[1],), activation='relu')) # layer 1
model3.add(Dense(10, activation='relu')) # layer 2
model3.add(Dense(1, activation='sigmoid')) # layer 3
# Compile the model
model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train the model
history3 = model3.fit(X_train, y_train, epochs=2000, validation_split=0.2, verbose=0)
```

Todo 11: Plot the training and validation loss for model 3 with iterations.

```
plot_history(history3, 'loss')
```



***Todo 12:** **

predict the values for test data and build the confusion matrix.

Comment on model performance.

```
y_pred = model3.predict(X_test)
y_pred = np.round(y_pred)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
print(classification_report(y_test, y_pred), '\n')
```