HOSPITAL RATING CLASSIFICATION

Welcome to the Starter Code for the Hospital Rating Classification Capstone Project!

In this notebook you'll find

- A blueprint on how to attempt the course project.
- Additional hints and directions on different tasks

Please note that this approach is one of the many approaches you can take for solving this Capstone project.

Import the necessary libraries

In [1]:
```
import pandas as pd, numpy as np
import seaborn as sns

from matplotlib import pyplot as plt
from pandas.core.common import random_state

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score, precision_score, recall_score, acc
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

from sklearn.linear_model import Lasso, LassoCV, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn import tree

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

from sklearn.metrics import r2_score
# Import 'MinMaxScaler' from 'sklearn'
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

Task 1

Task 1: Understand the data

Take some time to familiarize yourself with the data. What are the key variables?

Specifically, answer the following questions:

- 1.1 - Perform a few basic data quality checks to understand the different columns and prepare descriptive statistics for some of the important columns.
- 1.2 - What is the distribution of hospital overall ratings? How are they varying across other parameters like `State` ? Create a few visualizations that provide some insights

Task 1.1

In [3]:
```
df = pd.read_csv('hospital-info.csv')      ## Write the code to load the
```

In [4]:
```
df.head()
```

Out[4]:

	Provider ID	Hospital Name	Address	City	State	ZIP Code	County Name	Phc Numl
0	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	HOUSTON	3347938
1	10005	MARSHALL MEDICAL CENTER SOUTH	2505 U S HIGHWAY 431 NORTH	BOAZ	AL	35957	MARSHALL	2565938
2	10006	ELIZA COFFEE MEMORIAL HOSPITAL	205 MARENGO STREET	FLORENCE	AL	35631	LAUDERDALE	2567688
3	10007	MIZELL MEMORIAL HOSPITAL	702 N MAIN ST	OPP	AL	36467	COVINGTON	3344933
4	10008	CRENSHAW COMMUNITY HOSPITAL	101 HOSPITAL CIRCLE	LUVERNE	AL	36049	CRENSHAW	3343353

5 rows × 94 columns

In [5]:

```
##Check the data type of the different columns
## Hint - You can use the .info() method here
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3057 entries, 0 to 3056
Data columns (total 94 columns):
 #   Column                                                  Non-Null Co
unt  Dtype
---  ------                                                  -----------
---  -----
 0   Provider ID                                             3057 non-nu
ll   int64
 1   Hospital Name                                           3057 non-nu
ll   object
 2   Address                                                 3057 non-nu
ll   object
 3   City                                                    3057 non-nu
ll   object
 4   State                                                   3057 non-nu
ll   object
 5   ZIP Code                                                3057 non-nu
ll   int64
 6   County Name                                             3057 non-nu
ll   object
 7   Phone Number                                            3057 non-nu
ll   int64
 8   Hospital Ownership                                      3057 non-nu
ll   object
 9   Emergency Services                                      3057 non-nu
ll   object
 10  Hospital overall rating                                 3057 non-nu
ll   int64
 11  Mortality national comparison                           3057 non-nu
ll   int64
 12  Safety of care national comparison                      3057 non-nu
ll   int64
 13  Readmission national comparison                         3057 non-nu
ll   int64
 14  Patient experience national comparison                  3057 non-nu
ll   int64
 15  Effectiveness of care national comparison               3057 non-nu
ll   int64
 16  Timeliness of care national comparison                  3057 non-nu
ll   int64
 17  Efficient use of medical imaging national comparison    3057 non-nu
ll   int64
 18  MORT_30_AMI_Score                                       3057 non-nu
ll   float64
 19  MORT_30_CABG_Score                                      3057 non-nu
ll   float64
 20  MORT_30_COPD_Score                                      3057 non-nu
ll   float64
 21  MORT_30_HF_Score                                        3057 non-nu
ll   float64
 22  MORT_30_PN_Score                                        3057 non-nu
ll   float64
 23  MORT_30_STK_Score                                       3057 non-nu
ll   float64
 24  rating_group                                            3057 non-nu
ll   float64
 25  READM_30_AMI_Score                                      3057 non-nu
ll   float64
 26  READM_30_CABG_Score                                     3057 non-nu
ll   float64
```

```
 27  READM_30_COPD_Score                                3057 non-nu
ll   float64
 28  READM_30_HF_Score                                  3057 non-nu
ll   float64
 29  READM_30_HIP_KNEE_Score                            3057 non-nu
ll   float64
 30  READM_30_HOSP_WIDE_Score                           3057 non-nu
ll   float64
 31  READM_30_PN_Score                                  3057 non-nu
ll   float64
 32  READM_30_STK_Score                                 3057 non-nu
ll   float64
 33  TIME_OP_21_Score                                   3057 non-nu
ll   float64
 34  TIME_OP_5_Score                                    3057 non-nu
ll   float64
 35  EFF_EDV_Score                                      3057 non-nu
ll   float64
 36  EFF_ED_1b_Score                                    3057 non-nu
ll   float64
 37  EFF_ED_2b_Score                                    3057 non-nu
ll   float64
 38  EFF_IMM_2_Score                                    3057 non-nu
ll   float64
 39  EFF_IMM_3_OP_27_FAC_ADHPCT_Score                   3057 non-nu
ll   float64
 40  EFF_OP_18b_Score                                   3057 non-nu
ll   float64
 41  EFF_OP_20_Score                                    3057 non-nu
ll   float64
 42  EFF_OP_22_Score                                    3057 non-nu
ll   float64
 43  EFF_OP_29_Score                                    3057 non-nu
ll   float64
 44  EFF_OP_30_Score                                    3057 non-nu
ll   float64
 45  EFF_OP_4_Score                                     3057 non-nu
ll   float64
 46  EFF_PC_01_Score                                    3057 non-nu
ll   float64
 47  EFF_STK_1_Score                                    3057 non-nu
ll   float64
 48  EFF_STK_10_Score                                   3057 non-nu
ll   float64
 49  EFF_STK_2_Score                                    3057 non-nu
ll   float64
 50  EFF_STK_4_Score                                    3057 non-nu
ll   float64
 51  EFF_STK_5_Score                                    3057 non-nu
ll   float64
 52  EFF_STK_6_Score                                    3057 non-nu
ll   float64
 53  EFF_VTE_1_Score                                    3057 non-nu
ll   float64
 54  EFF_VTE_2_Score                                    3057 non-nu
ll   float64
 55  EFF_VTE_3_Score                                    3057 non-nu
ll   float64
 56  EFF_VTE_5_Score                                    3057 non-nu
ll   float64
 57  EFF_VTE_6_Score                                    3057 non-nu
```

ll   float64
 58 EXP_H_CLEAN_STAR_RATING_Score   3057 non-nu
ll   float64
 59 EXP_H_COMP_1_STAR_RATING_Score   3057 non-nu
ll   float64
 60 EXP_H_COMP_2_STAR_RATING_Score   3057 non-nu
ll   float64
 61 EXP_H_COMP_3_STAR_RATING_Score   3057 non-nu
ll   float64
 62 EXP_H_COMP_4_STAR_RATING_Score   3057 non-nu
ll   float64
 63 EXP_H_COMP_5_STAR_RATING_Score   3057 non-nu
ll   float64
 64 EXP_H_COMP_6_STAR_RATING_Score   3057 non-nu
ll   float64
 65 EXP_H_COMP_7_STAR_RATING_Score   3057 non-nu
ll   float64
 66 EXP_H_HSP_RATING_STAR_RATING_Score   3057 non-nu
ll   float64
 67 EXP_H_QUIET_STAR_RATING_Score   3057 non-nu
ll   float64
 68 EXP_H_RECMND_STAR_RATING_Score   3057 non-nu
ll   float64
 69 EXP_H_STAR_RATING_Score   3057 non-nu
ll   float64
 70 SAFETY_COMP_HIP_KNEE_Score   3057 non-nu
ll   float64
 71 SAFETY_PSI_12_POSTOP_PULMEMB_DVT_Score   3057 non-nu
ll   float64
 72 SAFETY_PSI_13_POST_SEPSIS_Score   3057 non-nu
ll   float64
 73 SAFETY_PSI_14_POSTOP_DEHIS_Score   3057 non-nu
ll   float64
 74 SAFETY_PSI_15_ACC_LAC_Score   3057 non-nu
ll   float64
 75 SAFETY_PSI_3_ULCER_Score   3057 non-nu
ll   float64
 76 SAFETY_PSI_4_SURG_COMP_Score   3057 non-nu
ll   float64
 77 SAFETY_PSI_6_IAT_PTX_Score   3057 non-nu
ll   float64
 78 SAFETY_PSI_7_CVCBI_Score   3057 non-nu
ll   float64
 79 SAFETY_PSI_90_SAFETY_Score   3057 non-nu
ll   float64
 80 SAFETY_HAI_1_SIR_Score   3057 non-nu
ll   float64
 81 SAFETY_HAI_1a_SIR_Score   3057 non-nu
ll   float64
 82 SAFETY_HAI_2_SIR_Score   3057 non-nu
ll   float64
 83 SAFETY_HAI_2a_SIR_Score   3057 non-nu
ll   float64
 84 SAFETY_HAI_3_SIR_Score   3057 non-nu
ll   float64
 85 SAFETY_HAI_4_SIR_Score   3057 non-nu
ll   float64
 86 SAFETY_HAI_5_SIR_Score   3057 non-nu
ll   float64
 87 SAFETY_HAI_6_SIR_Score   3057 non-nu
ll   float64

```
 88  MED_OP_10_Score                                        3057 non-nu
ll   float64
 89  MED_OP_11_Score                                        3057 non-nu
ll   float64
 90  MED_OP_13_Score                                        3057 non-nu
ll   float64
 91  MED_OP_14_Score                                        3057 non-nu
ll   float64
 92  MED_OP_8_Score                                         3057 non-nu
ll   float64
 93  MED_OP_9_Score                                         3057 non-nu
ll   float64
dtypes: float64(76), int64(11), object(7)
memory usage: 2.2+ MB
```

Note down your observations after completing the above task. You should ask questions such as:

- Check for non-null columns. Do you see any column having missing values?
- Are the datatypes correct for all the variables? You might have to convert a few of them to categorical later

> There are no missing or null value in given dataset.

> Yes, some variables have incorrect datatype. I will convert a few of them to categorical when it needed.

Task 1.2

In [6]:
```
## Descriptive Statistics
## Let's take the main measures and the hospital overall rating first.
features = ['Mortality national comparison', 'Safety of care national co
            'Readmission national comparison', 'Patient experience natio
            'Effectiveness of care national comparison', 'Timeliness of
            'Efficient use of medical imaging national comparison']
```

In [7]:
```
#making dataframe of main measures
new_df = df[['Hospital overall rating','Mortality national comparison',
         'Safety of care national comparison','Readmission national comp
         'Patient experience national comparison','Effectiveness of care
         'Timeliness of care national comparison','Efficient use of medi
```

In [8]:
```
### Filter out the above columns from the DataFrame and compute the desc
### Hint - The .describe() method might be useful
filtered_df=df.drop(labels=None, axis=0, index=None, columns=features, 1
```

In [9]:
```
filtered_df.describe()       #descriptive statistics
```

Out[9]:

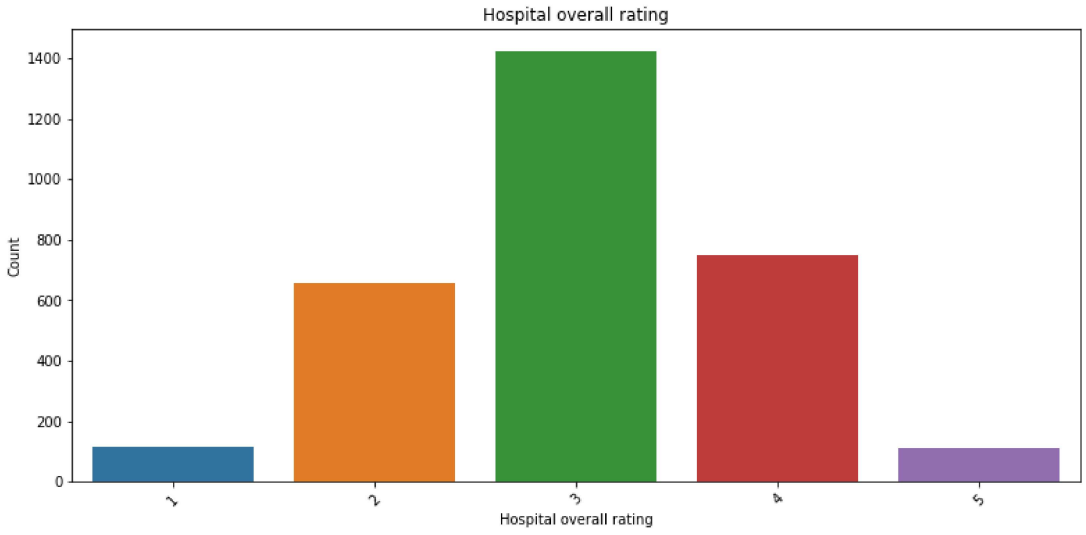
	Provider ID	ZIP Code	Phone Number	Hospital overall rating	MORT_30_AMI_Score	M
count	3057.000000	3057.000000	3.057000e+03	3057.000000	3057.000000	
mean	261817.891397	51757.296696	5.913678e+09	3.025842	-14.052018	
std	159800.930561	27935.174759	2.383292e+09	0.869600	1.096588	
min	10001.000000	613.000000	9.369338e+08	1.000000	-20.000000	
25%	110089.000000	30223.000000	3.607345e+09	2.000000	-14.500000	
50%	260022.000000	48617.000000	6.072744e+09	3.000000	-14.054091	
75%	390142.000000	76017.000000	8.059556e+09	4.000000	-13.400000	
max	670098.000000	99801.000000	9.898943e+09	5.000000	-9.400000	

8 rows × 80 columns

In [10]:
```
### Analyze your target variable - "Hospital overall rating"
### How does the ratings distribution look like for all the hospitals?

## Hint - You can use seaborn plots like countplot() for checking distri
## Hint - You can plot a correlation heatmap to check the correlation be
## Hint - You can also check the correlations between the "Hospital over
```

In [11]:
```
# countplot of hospital overall rating
plt.figure(figsize = (13, 6))
sns.countplot(x = 'Hospital overall rating', data = filtered_df)
xt = plt.xticks(rotation=45)
plt.xlabel('Hospital overall rating')
plt.ylabel('Count')
plt.title('Hospital overall rating')
```

Out[11]:
```
Text(0.5, 1.0, 'Hospital overall rating')
```



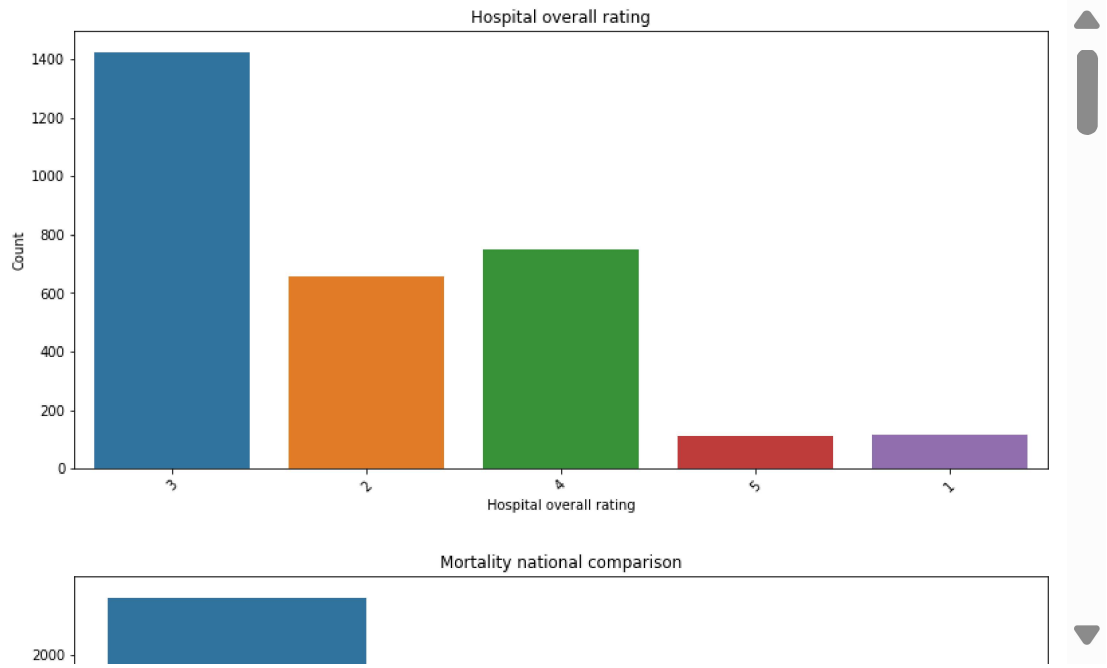
In [12]:

```
import pandas as pd
new_df2 = df[['Hospital overall rating','Mortality national comparison',
              'Readmission national comparison', 'Patient experience natio
              'Effectiveness of care national comparison', 'Timeliness of
              'Efficient use of medical imaging national comparison']].cop
new_df2 = new_df2.astype(str)
print(new_df2.dtypes)                          #converting the datatypes of
```

```
Hospital overall rating                                  object
Mortality national comparison                            object
Safety of care national comparison                       object
Readmission national comparison                          object
Patient experience national comparison                   object
Effectiveness of care national comparison                object
Timeliness of care national comparison                   object
Efficient use of medical imaging national comparison     object
dtype: object
```

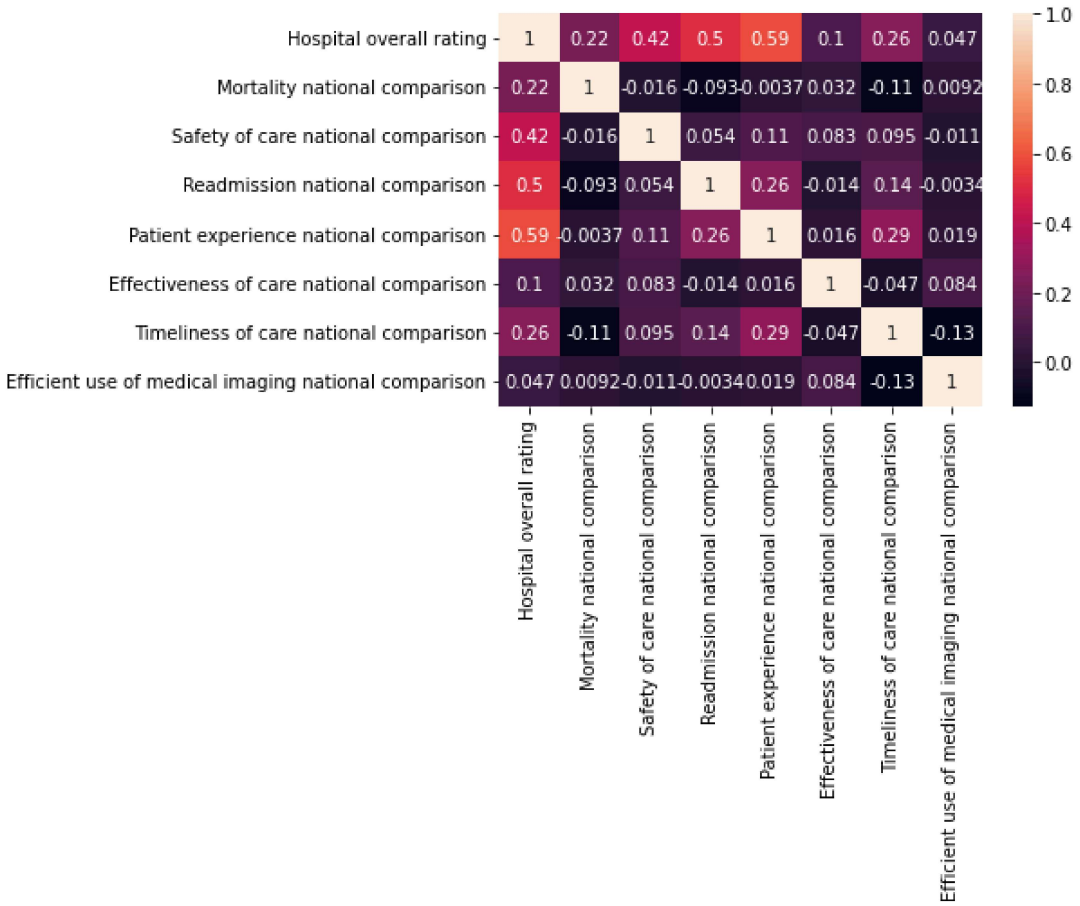
In [13]:

```
features = ['Hospital overall rating','Mortality national comparison',\
          'Safety of care national comparison','Readmission national comp
          'Patient experience national comparison','Effectiveness of care
          'Timeliness of care national comparison','Efficient use of medi

for i, item in enumerate(features):
    plt.figure(figsize = (13, 6))
    sns.countplot(x = item, data = new_df2)
    xt = plt.xticks(rotation=45)
    plt.xlabel(item)
    plt.ylabel('Count')
    plt.title(item)                            #visualisation of main measu
```



In [14]:
```
sns.heatmap(df[features].corr(),annot=True)                    # a correlation
```

Out[14]: <Axes: >



In [15]:
```
#correlations between the "Hospital overall rating" and other measures
corr = new_df.corr()
corr
```

Out[15]:

	Hospital overall rating	Mortality national comparison	Safety of care national comparison	Readmission national comparison	Patient experience national comparison	Effectiveness of ca nation comparisc
Hospital overall rating	1.000000	0.222774	0.421701	0.503575	0.586112	0.1010(
Mortality national comparison	0.222774	1.000000	-0.015789	-0.092562	-0.003655	0.0316
Safety of care national comparison	0.421701	-0.015789	1.000000	0.053738	0.110420	0.0829(
Readmission national comparison	0.503575	-0.092562	0.053738	1.000000	0.263206	-0.0139(
Patient experience national comparison	0.586112	-0.003655	0.110420	0.263206	1.000000	0.0162(
Effectiveness of care national comparison	0.101060	0.031659	0.082909	-0.013965	0.016203	1.0000(
Timeliness of care national comparison	0.263083	-0.110332	0.094908	0.140957	0.292600	-0.0472
Efficient use of medical imaging national comparison	0.046826	0.009227	-0.010664	-0.003414	0.018990	0.0839(

In [16]:
```
### Check how the hospital ratings vary across other parameters
### Hint - Some example parameters are "State" and "Hospital Ownership"
### Hint - You can use the pivot_table functionality of pandas to perfor
```

In [17]:
```
#checkig how the hospital rating vary across state parameter and hospito
pivot = df.pivot_table(index=['Hospital overall rating', 'State', 'Hospi
pivot
```

Out[17]:

Hospital overall rating	State	Hospital Ownership	EFF_EDV_Score	EFF_ED_1b_Score	EFF_ED_2b_Score	EFI
1	AR	Government	-2.000000	-394.000000	-268.000000	
		Voluntary	-2.000000	-240.000000	-170.000000	
	AZ	Others	-1.000000	-344.000000	-158.000000	
		Voluntary	-1.000000	-336.000000	-123.000000	
	CA	Government	-2.400000	-460.400000	-202.600000	
...	...	...	...	...	...	
5	TX	Others	-0.250000	-238.298843	-89.048859	
		Voluntary	-1.444444	-274.620679	-101.509590	
	WA	Government	-2.000000	-248.000000	-96.000000	
	WI	Others	-1.374185	-290.195371	-110.195438	
		Voluntary	-1.274837	-222.439074	-78.039088	

435 rows × 86 columns

Note down your observations after completing the above task. You should ask questions such as:

- How are ratings distributed? the Are you seeing any peculiar distributions for the ratings?
- How do the correlations between the measures and the target variable look like?
- How do ratings vary across the different levels of the parameter that you have taken?

Hospital rating is given from 1 to 5. Maximum 3 rating is observed and minimum 1 rating is observed among all the hospitals.

Mostly all variables have positive co relation with target variable.

Ratings are distributed among all types of hospitals. There is no specific category observed in low rating or in high rating.

Task 2 - Building machine learning models

Use your knowledge of classification models to create three models that predict hospital ratings. You should follow these steps:

- Prepare the data for the machine learning model
  - Remove all the demographic columns as well as any other uneccessary features from the data set
  - For simplification, instead of having 5 ratings, we will convert them to 0 and 1. Here 0 indicates that the hospital has been rated 3 or below and 1 indicates that the hospital has been rated as 4 or 5. Encode the Hospital columns as follows

    ```
    1,2,3 : 0
    4,5: 1
    ```

  - Store the predictors and the target variable in variables X and y.
  - Create the dummy variables for categorical columns.
  - Split the data into train and test sets (70-30 split with random state 0. This random state is recommended, though you can use any other random state of your choice).
  - Scale the numerical columns using StandardScaler.
- Build 3 classification models on your dataset. Carefully apply regularization and hyperparameter tuning techniques to improve your model performance for each of the models.
- Summarize the classification performance in terms of the necessary metrics such as accuracy, sensitivity, specificity, etc.

Prepare the data for machine learning model

In [18]:
```
## Drop all the demographic features
demo_features = ['Provider ID','Hospital Name',
 'Address',
 'City',
 'State',
 'ZIP Code',
 'County Name',
 'Phone Number']
```

In [19]:
```
## Drop all the above features from the DataFrame df and store the rest
df2 = df.drop(labels=None, axis=0, index=None, columns=demo_features, le
```

In [20]:
```
### Check the first 5 rows of df2 to see if the drop operation has worke
df2.head()
```

Out[20]:

	Hospital Ownership	Emergency Services	Hospital overall rating	Mortality national comparison	Safety of care national comparison	Readmission national comparison	Patient experience nationa comparisor
0	Government	Yes	3	1	2	1	0
1	Government	Yes	3	0	1	2	1
2	Government	Yes	2	0	1	1	0
3	Voluntary	Yes	3	1	1	0	1
4	Others	Yes	3	1	1	1	1

5 rows × 86 columns

In [21]:
```
##Recheck the columns to see if anything else needs to be dropped
## There might be other unnecessary columns that require dropping
```

There is no any unnecessary columns that required dropping.

Map the ratings

- 1,2,3 will be 0
- 4,5 will be 1

In [22]:
```
## Hint -  Write a simple lambda function to do the mapping
## Refer to this link from Course 1 for more help -  https://learn.upgrd
```

In [23]:
```
new_df1 = df2[['Hospital overall rating']].copy()       # making a new s
```

In [24]:
```
#using lambda function to convert the rating in o or 1.
df4=new_df1[['Hospital overall rating']].apply(lambda x: 0 if x['Hospita
df6 = pd.concat([new_df1,df4], axis=1)
df6.rename(columns = {'o':'value'}, inplace = True)
df6
```

Out[24]:

	Hospital overall rating	0
0	3	0
1	3	0
2	2	0
3	3	0
4	3	0
...	...	...
3052	4	1
3053	3	0
3054	3	0
3055	3	0
3056	3	0

3057 rows × 2 columns

Convert the datatypes of the categorical variables

In [25]:
```
### In task 1, you would have identified the categorical variables, whic
### Now is the right time to convert them to the correct datatype
### This will be useful when you create dummy variables next
```

In [26]:
```
new_df2 = df[['Hospital overall rating','Mortality national comparison',
              'Readmission national comparison', 'Patient experience natic
              'Effectiveness of care national comparison', 'Timeliness of
              'Efficient use of medical imaging national comparison']].cop
new_df2 = new_df2.astype(str)
print(new_df2.dtypes)                          #converting the datatypes
```

```
Hospital overall rating                                  object
Mortality national comparison                            object
Safety of care national comparison                       object
Readmission national comparison                          object
Patient experience national comparison                   object
Effectiveness of care national comparison                object
Timeliness of care national comparison                   object
Efficient use of medical imaging national comparison     object
dtype: object
```

Data Preparation and Train-test split

In [27]:
```
### Create X and y variable
X = df.drop(columns=['Hospital overall rating'])
y = df['Hospital overall rating']
```

In [28]:
```
### Create the dummy variables for categorical variables
### Note - Make sure the "drop_first parameter" is correctly initialized
### Hint - You can create multiple versions of the X dataset
X = pd.get_dummies(X, drop_first=False) #for kNN and trees
X2 = pd.get_dummies(X, drop_first=True) #for regression
```

In [29]:
```
## Perform the train_test split to create the train and validation sets
## Choose any random state of your choice
## Split it in the ratio of 70-30
X_train, X_val, X2_train, X2_val, y_train, y_val = train_test_split(X, )
```

In [30]:
```
# Scale and Standardize the numerical variables
scaler = StandardScaler()
numeric_cols = [col for col in X.columns if X[col].dtypes != 'object']

X_train[numeric_cols]= scaler.fit_transform(X_train[numeric_cols])
X_val[numeric_cols]= scaler.transform(X_val[numeric_cols])
```

In [31]:
```
X_scaled = scaler.fit_transform(X)
X2_train, X2_val, y_train, y_val = train_test_split(X_scaled, y, test_si
```

Model building and evaluation

You have learned multiple classification models till now, such as logistic regression, k-NN and decision trees. You should choose 3 of the models for performing the tasks in this section. You can follow the below steps:

- Build the models and fit it on training data
- Perform classifications on the validation data
- Compute and tabulate the validation accuracies for the different models
- Compare the accuracies for the different models and choose the best model

Note - You can also evaluate your models using additional metrics like `F1 score`, `Sensitivity`, `Specificity`, etc.

Helpful Resource - For writing precise code for this section, you can refer to the code you learned in Model Selection Lab Session in the `kNN and Model Selection` module.

- Additional notes
  - You can peform additional tasks like building ROC/AUC curves for all the models and identifying an optimal cut-off
  - You can also build conjectures around some arbitrary metric cut-offs. For example, say you want to build a model which has atleast 50% accuracy, specificity and sensitivity. Use these conjectures to arrive at a final model
  - Note that there is no right answer for this particular question. You will be awarded marks as long as your overall approach is correct

Simple linear model

In [32]:
```
#building a regression model
regression_model = LinearRegression()
regression_model.fit(X2_train, y_train)

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_mod
```

```
The coefficient for Provider ID is 0.005494760604615824
The coefficient for ZIP Code is 0.0034964418641713936
The coefficient for Phone Number is -0.0008788170737462873
The coefficient for Mortality national comparison is 0.0258530626336
72316
The coefficient for Safety of care national comparison is 0.03814551
152261679
The coefficient for Readmission national comparison is 0.03826857399
827246
The coefficient for Patient experience national comparison is 0.0305
3166295345591
The coefficient for Effectiveness of care national comparison is 0.0
076270789478263025
The coefficient for Timeliness of care national comparison is 0.0036
696126476568284
The coefficient for Efficient use of medical imaging national compar
ison is 0.005199167481442969
The coefficient for MORT_30_AMI_Score is 0.015892306215804987
The coefficient for MORT_30_CABG_Score is 0.005548079252956585
```

Here the coeficient values are relatively smaller. So we can say this is the smoother model.

In [33]:
```
intercept = regression_model.intercept_

print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is 3.021913081358539
```

Regularized Ridge Model

In [34]:
```
ridge = Ridge(alpha=.3) #coefficients are prevented to become too big by
ridge.fit(X2_train,y_train)
for i,col in enumerate(X_train.columns):
    print ("Ridge model coefficients for {} is {}:".format(col,ridge.coe
```

```
Ridge model coefficients for Provider ID is 0.0028218546777609315:
Ridge model coefficients for ZIP Code is 0.005702856281768828:
Ridge model coefficients for Phone Number is -0.002846358138023944:
Ridge model coefficients for Mortality national comparison is 0.0278
98957623848038:
Ridge model coefficients for Safety of care national comparison is
0.03999160203833633:
Ridge model coefficients for Readmission national comparison is 0.03
680335790538791:
Ridge model coefficients for Patient experience national comparison
is 0.02738377391865501:
Ridge model coefficients for Effectiveness of care national comparis
on is 0.006112095287492223:
Ridge model coefficients for Timeliness of care national comparison
is 0.009545951698017564:
Ridge model coefficients for Efficient use of medical imaging nation
al comparison is 0.006600342641302576:
Ridge model coefficients for MORT_30_AMI_Score is 0.0165280412324824
6:
```

We can see less coefficients values compared to linear regression. Since it is not a smoother model we will see difference in coefficient.

Regularized LASSO Model

In [35]:
```
lasso = Lasso(alpha=0.1)
lasso.fit(X2_train,y_train)
for i,col in enumerate(X2_train):
    print ("Lasso model coefficients for {} is {}:".format(col,lasso.coe
```

```
Lasso model coefficients for [-1.07535043 -1.13675648 -1.63071504
...  0.77709876 -0.19044996
  0.19044996] is 0.0:
Lasso model coefficients for [ 1.18310346  0.92091464 -1.30038182
...  0.77709876 -0.19044996
  0.19044996] is 0.0:
Lasso model coefficients for [-0.69979682 -0.19495806 -1.56101684
...  0.77709876 -0.19044996
  0.19044996] is -0.0:
Lasso model coefficients for [-1.32267355  1.53196477 -0.33921087
...  0.77709876 -0.19044996
  0.19044996] is 0.0:
Lasso model coefficients for [-0.69954647 -0.15693526  0.73278516
...  0.77709876 -0.19044996
  0.19044996] is 0.0:
Lasso model coefficients for [-0.76180285  0.31122502  0.94244123
...  0.77709876 -0.19044996
  0.19044996] is 0.0:
Lasso model coefficients for [ 1.4288744  -0.98301832 -0.21139326
```

Many of the coefficients have become 0 so we can drop of those dimensions from the model.It has taken only 5 dimensions to build the model.Lasso is also used for feature selection.

Comparing the scores

In [36]:
```
print(regression_model.score(X2_train, y_train))
print(regression_model.score(X2_val, y_val))
```

```
1.0
0.7710263419135246
```

In [37]:
```
print(ridge.score(X2_train, y_train))
print(ridge.score(X2_val, y_val))
```

```
0.999999999652323
0.7725581178846609
```

Accuracy of linear and ridge are more or less same because both coefficients values are similar

In [38]:
```
print(lasso.score(X2_train, y_train))
print(lasso.score(X2_val, y_val))
```

```
0.9060152462983453
0.9058277018248577
```

In [39]:
```
# Linear model - USE LassoCV to get the best LASSO model

from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV

lasso_model = Lasso(alpha=1)
lasso_model.fit(X2_train, y_train)
```

Out[39]:
```
Lasso(alpha=1)
```

In [40]:
```
#rmse function
def rmse(y_train, y_pred):
  return np.sqrt(mean_squared_error(y_train, y_pred))
```

In [41]:
```
print('RMSE training set', round(rmse(y_train, lasso_model.predict(X2_tr
print('RMSE validation set', round(rmse(y_val, lasso_model.predict(X2_va
```

```
RMSE training set 0.9
RMSE validation set 0.9
```

In [42]:
```
coef_table = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(np.transpos
coef_table.columns = ['', 'Coefficient']
coef_table.set_index('', inplace=True)
coef_table
```

Out[42]:

	Coefficient
Provider ID	0.0
ZIP Code	0.0
Phone Number	-0.0
Mortality national comparison	0.0
Safety of care national comparison	0.0
...	...
Hospital Ownership_Government	-0.0
Hospital Ownership_Others	-0.0
Hospital Ownership_Voluntary	0.0
Emergency Services_No	0.0
Emergency Services_Yes	-0.0

9042 rows × 1 columns

In [43]:
```
alphas = np.arange(0.01, 50, .5) # Check all alphas from .01 to 10 in st
lasso_cv_model = LassoCV(alphas= alphas , cv=5, max_iter=50000)
# fit model
lasso_cv_model.fit(X2_train, y_train)
# summarize chosen configuration
print('alpha: %f' % lasso_cv_model.alpha_)
```

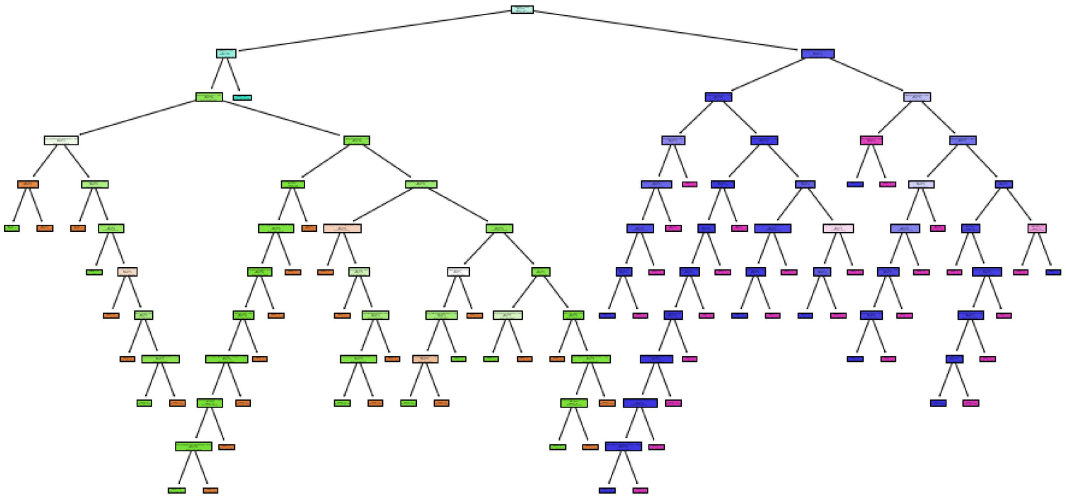
```
alpha: 0.010000
```

In [44]:
```
# Set best alpha
lasso_best_model = Lasso(alpha=lasso_cv_model.alpha_)
lasso_best_model.fit(X2_train, y_train)

print('Best Lasso-RMSE training set', round(rmse(y_train, lasso_best_mod
print('Best Lasso-RMSE validation set', round(rmse(y_val, lasso_best_mod
```

```
Best Lasso-RMSE training set 0.127
Best Lasso-RMSE validation set 0.212
```

In [45]:
```
# Tree Model - Use max depth to control the complexity of the tree. Run

tree_model_1 = DecisionTreeClassifier(random_state = 1)
tree_model_1 = tree_model_1.fit(X_train, y_train)
```

In [46]:
```
# Visualize the decision tree for 'tree_model_1'
# Note: This cell may take a while to run owing to the large number of r

fig = plt.figure(figsize = (16,8))
fig = tree.plot_tree(tree_model_1, feature_names = X.columns, filled = T
```



In [47]:
```
# Print the number of leaves and the depth of the tree for 'tree_model_1
tree_model_1_n_leaves = tree_model_1.get_n_leaves()
tree_model_1_depth = tree_model_1.get_depth()
print('Number of leaves =', tree_model_1_n_leaves)
print('Tree depth =', tree_model_1_depth)
```

```
Number of leaves = 65
Tree depth = 11
```

In [48]:
```
# Obtain predicted class labels for the training and validation data usi
# Hint: Study the documentation of the 'predict()' method
y_pred_1_train = tree_model_1.predict(X_train)
y_pred_1_val = tree_model_1.predict(X_val)
```

In [49]:
```
# Compute the accuracy and the sensitivity of 'tree_model_1' on the trai
# Note: The 'pos_label' parameter for the 'recall_score()' method should
# Note: The positive class label in this exercise is '1', which is also
acc_train_1 = accuracy_score(y_train, y_pred_1_train)
acc_val_1 = accuracy_score(y_val, y_pred_1_val)

# Summarize the above metrics for the train and validation sets using a
tree_model_1_metrics = pd.DataFrame(data = {'Accuracy': [acc_train_1, ac
                                                         },
                                            index = ['tree_model_1_train', 'tree

tree_model_1_metrics
```

Out[49]:

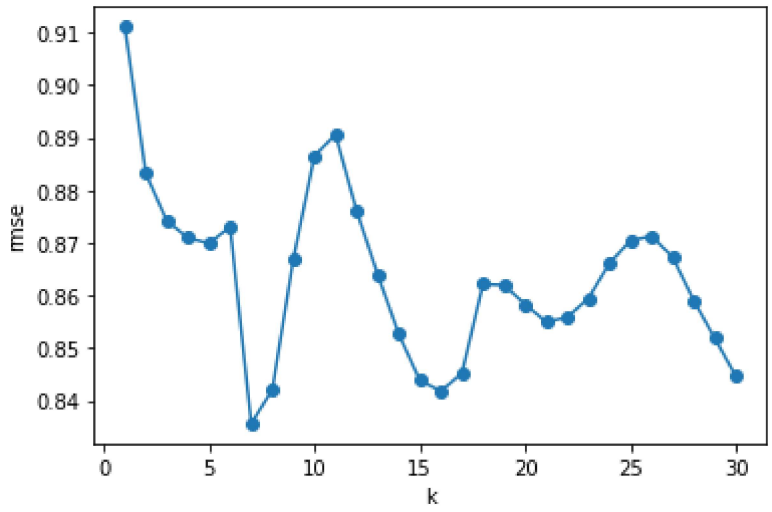
	Accuracy
tree_model_1_train	1.00000
tree_model_1_val	0.93573

In [50]:
```
y = df['Hospital overall rating']
y_train, y_val = train_test_split(y, test_size=0.3, random_state = 1)
```

In [51]:
```
#rmse function
def rmse(y_train, y_pred):
  return np.sqrt(mean_squared_error(y_train, y_pred))
```

In [52]:
```
# KNN Model

##### CODE HERE #####
X_train= scaler.fit_transform(X_train)
X_val= scaler.transform(X_val)



# Find the value of k for which RMSE is minimum, using GridSearchCV

##### CODE HERE #####
kvalues = np.arange(1,31) # Parameter range

val_rmse=[]

for k in kvalues:
  knn_reg = KNeighborsRegressor(n_neighbors=k)
  knn_reg.fit(X_train, y_train)
  y_pred = knn_reg.predict(X_val)
  val_rmse.append(rmse(y_val, y_pred))

# Find the value of k for which RMSE is minimum, using GridSearchCV

##### CODE HERE #####
### Rmse vs k
plt.plot(kvalues,val_rmse,marker='o')
plt.xlabel("k")
plt.ylabel("rmse")
print("The minimum rmse is obtained at k = " + str(np.argmin(val_rmse)+1
```

```
The minimum rmse is obtained at k = 7
```



In [53]:
```
knn_reg_best = KNeighborsClassifier(n_neighbors=29)
knn_reg_best.fit(X_train, y_train)
print('RMSE validation set:', round(rmse(y_val, knn_reg_best.predict(X_v
```

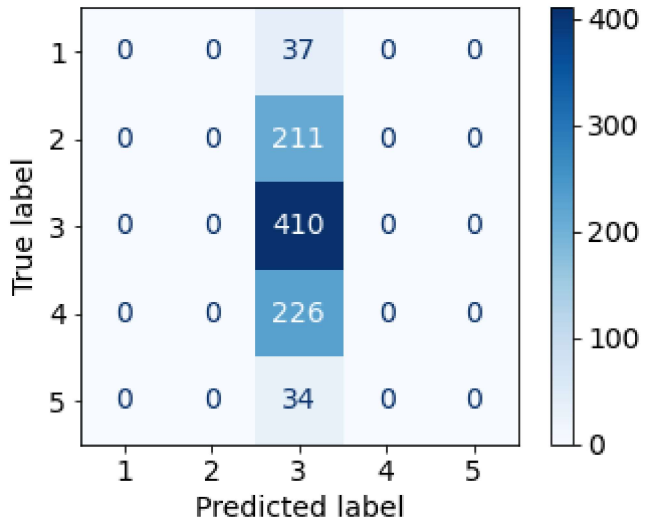
```
RMSE validation set: 0.89
```

In [54]:
```
##Let's fit the best model and calculate some classification performance
```

In [55]:
```
knn_clf_best = KNeighborsClassifier(n_neighbors=22)
knn_clf_best.fit(X_train, y_train)
y_pred = knn_clf_best.predict(X_val)
```

In [56]:
```
plt.rcParams.update({'font.size': 14}) # To make the plot labels easier
ConfusionMatrixDisplay.from_estimator(
        knn_clf_best,
        X_val,
        y_val,
        cmap=plt.cm.Blues,
    )
```

Out[56]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x25
480649880>
```



In [57]:
```
# Check all alphas from .01 to 10 in steps of size 0.25
alphas = np.arange(.01, 25, .25)
lasso_cv_model = LassoCV(alphas= alphas, cv=5, max_iter=50000)
lasso_cv_model.fit(X2_train, y_train)

# Train the lasso regression model with the best value of alpha
lin_reg_best = Lasso(alpha=lasso_cv_model.alpha_)
lin_reg_best.fit(X2_train, y_train)

# Calculate RMSE for the lasso regression model on train and validation
lin_train_rmse = rmse(y_train, lin_reg_best.predict(X2_train))
lin_val_rmse = rmse(y_val, lin_reg_best.predict(X2_val))
```

In [58]:

```
# Get the different values of ccp alphas
tree_reg = DecisionTreeRegressor()
path= tree_reg.cost_complexity_pruning_path(X_train,y_train)
ccp_alphas = path.ccp_alphas

# Train decision tree regressor models for different values of ccp_alpha

# Create a list to store the different tree models
regs=[]

# Iterate through different ccp alpha values and train models for each c
for ccp_alpha in ccp_alphas:
    # Create and train the model
    curr_reg = DecisionTreeRegressor(random_state=0, ccp_alpha = ccp_alp
    curr_reg.fit(X2_train,y_train)

# Save the model in the list
    regs.append(curr_reg)

# Calculate the RMSE for all the tree models

# Create lists to store RMSE on training and validation data sets
    train_rmse=[]
    val_rmse=[]

# Iterate through the models and calculate RMSE
for r in regs:
    y_train_pred=r.predict(X2_train)
    y_val_pred = r.predict(X2_val)

    train_rmse.append(rmse(y_train_pred,y_train))
    val_rmse.append(rmse(y_val_pred,y_val))

# Pick the best ccp alpha
    best_ccp_alpha = ccp_alphas[val_rmse.index(min(val_rmse))]

# Train the corresponding tree
    tree_reg_best = tree.DecisionTreeRegressor(random_state=0, ccp_alpha
    tree_reg_best.fit(X_train,y_train)

# Calculate RMSE for the best tree
    tree_train_rmse = rmse(y_train, tree_reg_best.predict(X_train))
    tree_val_rmse = rmse(y_val, tree_reg_best.predict(X_val))
```

In [59]:

```
# KNN Model (using our previous knowledge about the best k!)
knn_reg_best = KNeighborsRegressor(n_neighbors=16)
knn_reg_best.fit(X_train, y_train)

knn_train_rmse = rmse(y_train, knn_reg_best.predict(X_train))
knn_val_rmse = rmse(y_val, knn_reg_best.predict(X_val))
```

In [60]:
```
# Create a dataframe to display the RMSE values for the three models
pd.DataFrame([[lin_train_rmse, lin_val_rmse],[tree_train_rmse, tree_val_
              [knn_train_rmse, knn_val_rmse]], columns=['RMSE Train', 'R
              index = ['Linear', 'Tree', 'kNN'])
```

Out[60]:

	RMSE Train	RMSE Validation
Linear	0.127077	0.211775
Tree	0.169595	0.214776
kNN	0.775779	0.841843

In [61]:
```
# Perform train-validation split on the new independent variable
y_train, y_val = train_test_split(y, test_size=0.3, random_state = 1)
```

In [62]:
```
# Create and train the logistic regression model
log_clf_best = LogisticRegression(penalty='none', solver='lbfgs', random
                                  max_iter=200).fit(X2_train, y_train

# Calculate the logistic regression model's accuracy on the training and
log_train_acc = log_clf_best.score(X2_train, y_train)
log_val_acc = log_clf_best.score(X2_val, y_val)

# Create and train the decision tree classifier model
best_ccp_alpha = 0.004801587301587302 # from Module 4
tree_clf_best = DecisionTreeClassifier(random_state=0, ccp_alpha=best_cc
tree_clf_best.fit(X_train,y_train)

# Calculate the decision tree classifier model's accuracy on the trainin
tree_train_acc = tree_clf_best.score(X_train, y_train)
tree_val_acc = tree_clf_best.score(X_val, y_val)

# Create and train the kNN model
knn_clf_best = KNeighborsClassifier(n_neighbors=14)
knn_clf_best.fit(X_train, y_train)

# Calculate the kNN model's accuracy on the training and validation sets
knn_train_acc = knn_clf_best.score(X_train, y_train)
knn_val_acc = knn_clf_best.score(X_val, y_val)

# Create a dataframe to display the accuracy values for the three models
pd.DataFrame([[log_train_acc, log_val_acc], [tree_train_acc, tree_val_ac
              [knn_train_acc, knn_val_acc]], columns=['Training Acc', 'V
              index = ['Logistic', 'Tree', 'kNN'])
```

Out[62]:

	Training Acc	Validation Acc
Logistic	1.000000	0.371460
Tree	0.949509	0.937908
kNN	0.494156	0.448802

In [63]:

```
# Set potential cutoff values
clf_cutoffs = np.arange(0,1.01,0.01)

# Recover profit for training and testing data
profit_arr = np.array(df['Hospital overall rating'])
profit_train, profit_val = train_test_split(profit_arr, test_size=0.3, r

# For each model obtain the predicted probabilities and then save the to
table = []
for clf_model in [log_clf_best, tree_clf_best, knn_clf_best]:

    # In case of logistic regression, use X2
    if clf_model == log_clf_best:
      probs_train = clf_model.predict_proba(X2_train)[:, 1]
      probs_val = clf_model.predict_proba(X2_val)[:, 1]

    # In case of decision tree or kNN, use X
    else:
      probs_train = clf_model.predict_proba(X_train)[:, 1]
      probs_val = clf_model.predict_proba(X_val)[:, 1]

    # Add the profits for all models which satisfy the cutoff for differ
    clf_profits = []
    for cutoff in clf_cutoffs:
        clf_profits.append(sum(profit_train[probs_train > cutoff]))

    # Calculating the best cutoff
    best_profit_train = max(clf_profits)
    best_cutoff = clf_cutoffs[clf_profits.index(best_profit_train)]
    best_profit_val = sum(profit_val[probs_val > best_cutoff])
    table.append([best_cutoff, best_profit_train, best_profit_val])
```

In [64]:
```
# Set potential cutoff values
reg_cutoffs = np.arange(-5000, 2000)

# For each model obtain the predicted profit and then save the total pro
for reg_model in [lin_reg_best, tree_reg_best, knn_reg_best]:

    # In case of linear regression, use X2
    if reg_model == lin_reg_best:
      pred_train = reg_model.predict(X2_train)
      pred_val = reg_model.predict(X2_val)

    # In case of decision tree or kNN, use X
    else:
      pred_train = reg_model.predict(X_train)
      pred_val = reg_model.predict(X_val)

    # Add the profits for all models which satisfy the cutoff for differ
    reg_profits = []
    for cutoff in reg_cutoffs:
        reg_profits.append(sum(profit_train[pred_train > cutoff]))

    # Calculating the best cutoff
    best_profit_train = max(reg_profits)
    best_cutoff = reg_cutoffs[reg_profits.index(best_profit_train)]
    best_profit_val = sum(profit_val[pred_val > best_cutoff])
    table.append([best_cutoff, best_profit_train, best_profit_val])
```

In [65]:
```
table_df = pd.DataFrame(table, columns=['Cutoff Value', 'Train Profit',
                           index=['Logistic Clf','Tree Clf','KNN Clf','Line
table_df
```

Out[65]:

	Cutoff Value	Train Profit	Validation Profit
Logistic Clf	0.0	6487	2763
Tree Clf	0.0	970	459
KNN Clf	0.0	6487	2763
Linear Reg	-5000.0	6487	2763
Tree Reg	-5000.0	6487	2763
KNN Reg	-5000.0	6487	2763

Task 3

You have now built (at least) three machine learning models. Choose the best model according to your metrics and provide the following recommendations

- Hospital Rating Predictor: Using the best model of your choice, predict the ratings of a few new hospitals which are yet to be assigned a rating by CMS. The information for these hospitals has been provided in a separate CSV file named 'not_yet_rated.csv'.
- Hospital Improvement Plan: Let's say a few of the hospitals were rated low (0) by the model that you chose. Provide recommendations on how these hospitals can improve their ratings

In [66]:
```
###Let's read the not_yet_rated dataset
new = pd.read_csv('not_yet_rated.csv')
```

In [67]:
```
## Check the top 5 rows
new.head()
```

Out[67]:

	Provider ID	Hospital Ownership	Emergency Services	Mortality national comparison	Safety of care national comparison	Readmission national comparison	Patien experience nationa comparisor
0	520139	Voluntary	Yes	1	1	2	2
1	520189	Government	Yes	2	1	2	2
2	370029	Government	No	1	1	1	1
3	370032	Others	Yes	1	2	1	(
4	370036	Government	Yes	1	1	1	1

5 rows × 86 columns

Approach to predict ratings

- Perform the exact same data preparation steps as earlier
  - Drop the unnecessary columns
  - Convert the datatypes of categorical variables and create dummies
  - Standardize the numeric columns
- After that we shall use the `.predict()` method of your ML model to predict the ratings

In [68]:
```
new.info()                    #checking null values and data type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 86 columns):
 #   Column                                                Non-Null Co
unt  Dtype
---  ------                                                -----------
---  -----
 0   Provider ID                                           6 non-null
int64
 1   Hospital Ownership                                    6 non-null
object
 2   Emergency Services                                    6 non-null
object
 3   Mortality national comparison                         6 non-null
int64
 4   Safety of care national comparison                    6 non-null
int64
 5   Readmission national comparison                       6 non-null
int64
 6   Patient experience national comparison                6 non-null
int64
 7   Effectiveness of care national comparison             6 non-null
int64
 8   Timeliness of care national comparison                6 non-null
int64
 9   Efficient use of medical imaging national comparison  6 non-null
int64
 10  MORT_30_AMI_Score                                     6 non-null
float64
 11  MORT_30_CABG_Score                                    6 non-null
float64
 12  MORT_30_COPD_Score                                    6 non-null
float64
 13  MORT_30_HF_Score                                      6 non-null
float64
 14  MORT_30_PN_Score                                      6 non-null
float64
 15  MORT_30_STK_Score                                     6 non-null
float64
 16  rating_group                                          6 non-null
int64
 17  READM_30_AMI_Score                                    6 non-null
float64
 18  READM_30_CABG_Score                                   6 non-null
float64
 19  READM_30_COPD_Score                                   6 non-null
float64
 20  READM_30_HF_Score                                     6 non-null
float64
 21  READM_30_HIP_KNEE_Score                               6 non-null
float64
 22  READM_30_HOSP_WIDE_Score                              6 non-null
float64
 23  READM_30_PN_Score                                     6 non-null
float64
 24  READM_30_STK_Score                                    6 non-null
float64
 25  TIME_OP_21_Score                                      6 non-null
float64
 26  TIME_OP_5_Score                                       6 non-null
float64
```

```
 27  EFF_EDV_Score                                 6 non-null
int64
 28  EFF_ED_1b_Score                               6 non-null
int64
 29  EFF_ED_2b_Score                               6 non-null
int64
 30  EFF_IMM_2_Score                               6 non-null
int64
 31  EFF_IMM_3_OP_27_FAC_ADHPCT_Score              6 non-null
int64
 32  EFF_OP_18b_Score                              6 non-null
int64
 33  EFF_OP_20_Score                               6 non-null
int64
 34  EFF_OP_22_Score                               6 non-null
int64
 35  EFF_OP_29_Score                               6 non-null
float64
 36  EFF_OP_30_Score                               6 non-null
float64
 37  EFF_OP_4_Score                                6 non-null
float64
 38  EFF_PC_01_Score                               6 non-null
float64
 39  EFF_STK_1_Score                               6 non-null
float64
 40  EFF_STK_10_Score                              6 non-null
float64
 41  EFF_STK_2_Score                               6 non-null
float64
 42  EFF_STK_4_Score                               6 non-null
float64
 43  EFF_STK_5_Score                               6 non-null
float64
 44  EFF_STK_6_Score                               6 non-null
float64
 45  EFF_VTE_1_Score                               6 non-null
int64
 46  EFF_VTE_2_Score                               6 non-null
float64
 47  EFF_VTE_3_Score                               6 non-null
float64
 48  EFF_VTE_5_Score                               6 non-null
float64
 49  EFF_VTE_6_Score                               6 non-null
float64
 50  EXP_H_CLEAN_STAR_RATING_Score                 6 non-null
float64
 51  EXP_H_COMP_1_STAR_RATING_Score                6 non-null
float64
 52  EXP_H_COMP_2_STAR_RATING_Score                6 non-null
float64
 53  EXP_H_COMP_3_STAR_RATING_Score                6 non-null
float64
 54  EXP_H_COMP_4_STAR_RATING_Score                6 non-null
float64
 55  EXP_H_COMP_5_STAR_RATING_Score                6 non-null
float64
 56  EXP_H_COMP_6_STAR_RATING_Score                6 non-null
float64
 57  EXP_H_COMP_7_STAR_RATING_Score                6 non-null
```

```
float64
 58  EXP_H_HSP_RATING_STAR_RATING_Score                  6 non-null
float64
 59  EXP_H_QUIET_STAR_RATING_Score                       6 non-null
float64
 60  EXP_H_RECMND_STAR_RATING_Score                      6 non-null
float64
 61  EXP_H_STAR_RATING_Score                             6 non-null
float64
 62  SAFETY_COMP_HIP_KNEE_Score                          6 non-null
float64
 63  SAFETY_PSI_12_POSTOP_PULMEMB_DVT_Score              6 non-null
float64
 64  SAFETY_PSI_13_POST_SEPSIS_Score                     6 non-null
float64
 65  SAFETY_PSI_14_POSTOP_DEHIS_Score                    6 non-null
float64
 66  SAFETY_PSI_15_ACC_LAC_Score                         6 non-null
float64
 67  SAFETY_PSI_3_ULCER_Score                            6 non-null
float64
 68  SAFETY_PSI_4_SURG_COMP_Score                        6 non-null
float64
 69  SAFETY_PSI_6_IAT_PTX_Score                          6 non-null
float64
 70  SAFETY_PSI_7_CVCBI_Score                            6 non-null
float64
 71  SAFETY_PSI_90_SAFETY_Score                          6 non-null
float64
 72  SAFETY_HAI_1_SIR_Score                              6 non-null
float64
 73  SAFETY_HAI_1a_SIR_Score                             6 non-null
float64
 74  SAFETY_HAI_2_SIR_Score                              6 non-null
float64
 75  SAFETY_HAI_2a_SIR_Score                             6 non-null
float64
 76  SAFETY_HAI_3_SIR_Score                              6 non-null
float64
 77  SAFETY_HAI_4_SIR_Score                              6 non-null
float64
 78  SAFETY_HAI_5_SIR_Score                              6 non-null
float64
 79  SAFETY_HAI_6_SIR_Score                              6 non-null
float64
 80  MED_OP_10_Score                                     6 non-null
float64
 81  MED_OP_11_Score                                     6 non-null
float64
 82  MED_OP_13_Score                                     6 non-null
float64
 83  MED_OP_14_Score                                     6 non-null
float64
 84  MED_OP_8_Score                                      6 non-null
float64
 85  MED_OP_9_Score                                      6 non-null
float64
dtypes: float64(66), int64(18), object(2)
memory usage: 4.2+ KB
```

There is no null value in given data set. Some variables has incorrect data type. We will convert them when needed.

In [69]:
```
import pandas as pd
new_2 = new[[ 'Safety of care national comparison','Readmission national
            'Patient experience national comparison','Effectiveness of c
            'Timeliness of care national comparison',
            'Efficient use of medical imaging national comparison']].cop
new_2 = new_2.astype(str)
print(new_2.dtypes)           # convert the datatype into object
```

```
Safety of care national comparison                    object
Readmission national comparison                       object
Patient experience national comparison                object
Effectiveness of care national comparison             object
Timeliness of care national comparison                object
Efficient use of medical imaging national comparison  object
dtype: object
```

In [70]:
```
### Create X and y variable
X = new.drop(columns=['Provider ID'])
y = new['Provider ID']
```

In [71]:
```
### Create the dummy variables for categorical variables
### Note - Make sure the "drop_first parameter" is correctly initialized
### Hint - You can create multiple versions of the X dataset
X = pd.get_dummies(X, drop_first=False) #for kNN and trees
X2 = pd.get_dummies(X, drop_first=True) #for regression
```

In [72]:
```
## Perform the train_test split to create the train and validation sets
## Choose any random state of your choice
## Split it in the ratio of 70-30
X_train, X_val, X2_train, X2_val, y_train, y_val = train_test_split(X, X
```

In [73]:
```
# Scale and Standardize the numerical variables
scaler = StandardScaler()
numeric_cols1 = [col for col in X.columns if X[col].dtypes != 'object']

X_train[numeric_cols1]= scaler.fit_transform(X_train[numeric_cols1])
X_val[numeric_cols1]= scaler.transform(X_val[numeric_cols1])
```

In [74]:
```
X_scaled = scaler.fit_transform(X)
X2_train, X2_val, y_train, y_val = train_test_split(X_scaled, y, test_si
```

In [75]:
```
# Linear model - USE LassoCV to get the best LASSO model

from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV

lasso_model = Lasso(alpha=1)
lasso_model.fit(X2_train, y_train)
```

Out[75]:
```
Lasso(alpha=1)
```

In [76]:
```
#rmse function
def rmse(y_train, y_pred):
  return np.sqrt(mean_squared_error(y_train, y_pred))
```

In [77]:
```
print('RMSE training set', round(rmse(y_train, lasso_model.predict(X2_tr
print('RMSE validation set', round(rmse(y_val, lasso_model.predict(X2_va
```

```
RMSE training set 0.9
RMSE validation set 31185.8
```

In [78]:
```
coef_table = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(np.transpos
coef_table.columns = ['', 'Coefficient']
coef_table.set_index('', inplace=True)
coef_table.head(10)
```

Out[78]:

	Coefficient
Mortality national comparison	0.000000
Safety of care national comparison	-35378.849519
Readmission national comparison	25785.338487
Patient experience national comparison	12895.154512
Effectiveness of care national comparison	32232.213097
Timeliness of care national comparison	-26531.043732
Efficient use of medical imaging national comparison	-4661.708443
MORT_30_AMI_Score	2750.501747
MORT_30_CABG_Score	190.996795
MORT_30_COPD_Score	1064.759509

In [79]:
```
alphas = np.arange(0.01, 50, .5) # Check all alphas from .01 to 10 in st
lasso_cv_model = LassoCV(alphas= alphas , cv=3, max_iter=50000)
# fit model
lasso_cv_model.fit(X2_train, y_train)
# summarize chosen configuration
print('alpha: %f' % lasso_cv_model.alpha_)
```

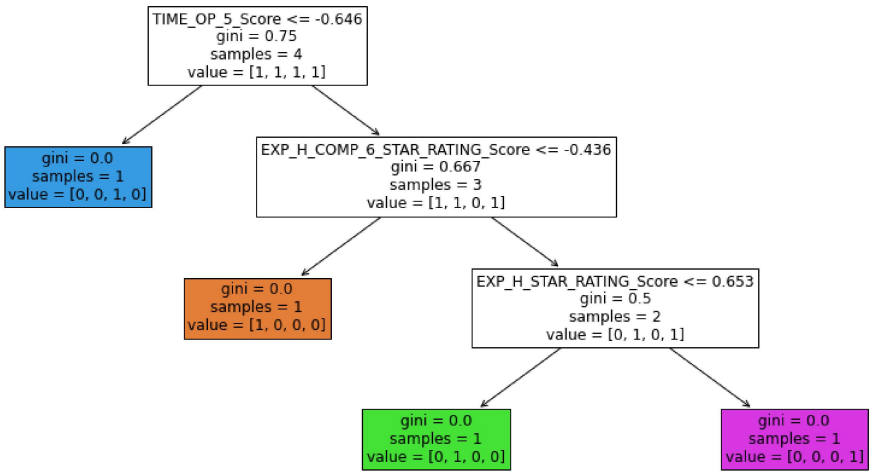
```
alpha: 49.510000
```

In [80]:
```
# Set best alpha
lasso_best_model = Lasso(alpha=lasso_cv_model.alpha_)
lasso_best_model.fit(X2_train, y_train)

print('Best Lasso-RMSE training set', round(rmse(y_train, lasso_best_mod
print('Best Lasso-RMSE validation set', round(rmse(y_val, lasso_best_mod
```

```
Best Lasso-RMSE training set 42.289
Best Lasso-RMSE validation set 30315.929
```

In [81]:
```
# Tree Model - Use max depth to control the complexity of the tree. Run

tree_model_2= DecisionTreeClassifier(random_state = 1)
tree_model_2 = tree_model_2.fit(X_train, y_train)
```

In [82]:
```
# Visualize the decision tree for 'tree_model_1'
# Note: This cell may take a while to run owing to the large number of r

fig = plt.figure(figsize = (16,8))
fig = tree.plot_tree(tree_model_2, feature_names = X.columns, filled = T
```



In [83]:
```
# Print the number of leaves and the depth of the tree for 'tree_model_1
tree_model_2_n_leaves = tree_model_2.get_n_leaves()
tree_model_2_depth = tree_model_2.get_depth()
print('Number of leaves =', tree_model_2_n_leaves)
print('Tree depth =', tree_model_2_depth)
```

```
Number of leaves = 4
Tree depth = 3
```

In [84]:
```
# Obtain predicted class labels for the training and validation data usi
# Hint: Study the documentation of the 'predict()' method
y_pred_1_train = tree_model_2.predict(X_train)
y_pred_1_val = tree_model_2.predict(X_val)
```

In [85]:
```
# Compute the accuracy and the sensitivity of 'tree_model_1' on the trai
# Note: The 'pos_label' parameter for the 'recall_score()' method should
# Note: The positive class label in this exercise is '1', which is also
acc_train_1 = accuracy_score(y_train, y_pred_1_train)
acc_val_1 = accuracy_score(y_val, y_pred_1_val)

# Summarize the above metrics for the train and validation sets using a
tree_model_1_metrics = pd.DataFrame(data = {'Accuracy': [acc_train_1, ac
                                            },
                                    index = ['tree_model_1_train', 'tree

tree_model_1_metrics
```

Out[85]:

	Accuracy
tree_model_1_train	1.0
tree_model_1_val	0.0

In [86]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_blobs
# generate 2d classification dataset
X, y = make_blobs(n_samples=3057, centers=2, n_features=2, random_state=
# fit final model
model = LogisticRegression()
model.fit(X, y)
# new instances where we do not know the answer
Xnew, _ = make_blobs(n_samples=6, centers=2, n_features=2, random_state=
# make a prediction
ynew = model.predict(Xnew)
# show the inputs and predicted outputs
for i in range(len(Xnew)):
    print("X=%s, Predicted=%s" % (Xnew[i], ynew[i]))          #predictio
```

```
X=[0.08525186 3.64528297], Predicted=0
X=[-2.18773166  3.33352125], Predicted=0
X=[-9.67867341 -4.20271892], Predicted=1
X=[-8.53560457 -6.01348926], Predicted=1
X=[-0.79415228  2.10495117], Predicted=0
X=[-10.32012971  -4.3374029 ], Predicted=1
```

Approach to identify areas of improvement

- Identify the measures which have a positive influence on the overall hospital ratings. For example,
  - if you're using a logistic regression model, you can check the coefficients
    - A +ve coefficient indicates +ve influence on the overall hospital rating
    - A -ve coefficient indicates -ve influence on the overall hospital rating
- Identify in which of the above measures a low-rated hospital is currently lagging behind. These measures need to be improved.
- Further deep dive into the sub-measures using the same approach as above.

> With the help of logistic regression model, we can say that there is mostly positive influence of measures on the overall hospital rating.

Safety of care national comparison, Efficient use of medical imaging national comparison, Timeliness of care national comparison measures need to be improved. Because of these measures a low rated hospital is currently lagging behind.

Many measures have more or less influence on rating. There is need of some improvemnt.