Problem Statement

To check which model is best and fit for this data

In [2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

Data collection

In [3]:
```
df=pd.read_csv(r"C:\Users\teppa\Downloads\insurance (2).csv")
df
```

Out[3]:

	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

Data cleaning

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

Out[4]:

	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

In [5]:
```
df.tail()
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

In [6]:
```
1 df.describe()
```

Out[6]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

In [7]:
```
1 df.isnull()
```

Out[7]:

	age	sex	bmi	children	smoker	region	charges
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False

1338 rows × 7 columns

Data visualization

In [8]:
```
1 convert={"sex":{"female":1,"male":0}}
2 df=df.replace(convert)
3 df
```

Out[8]:

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

1338 rows × 7 columns

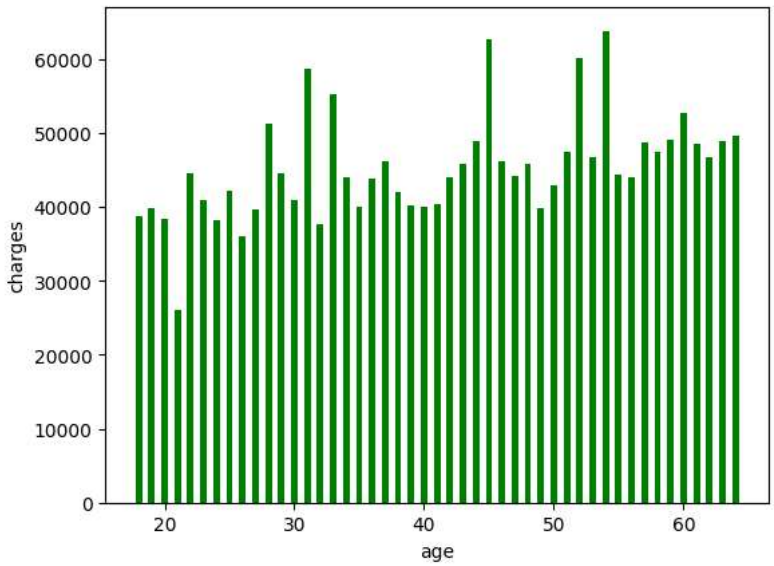
In [9]:
```
convert={"region":{"southwest":1,"southeast":2,"northeast":3,"northwest":4}}
df=df.replace(convert)
df
```

Out[9]:

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

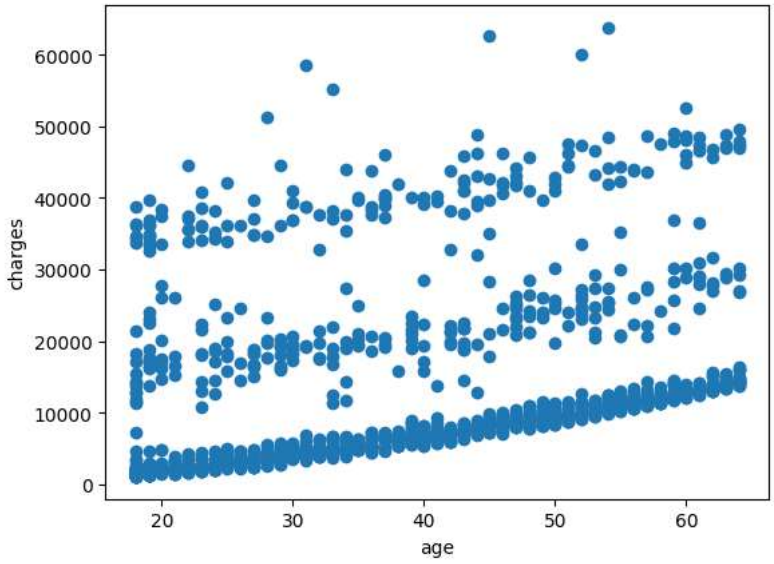
1338 rows × 7 columns

In [10]:
```
plt.bar(df['age'],df['charges'],label="age",color='b',width=0.5)
plt.bar(df['age'],df['charges'],label="charges",color='g',width=0.5)
plt.xlabel('age')
plt.ylabel('charges')
plt.show()
```

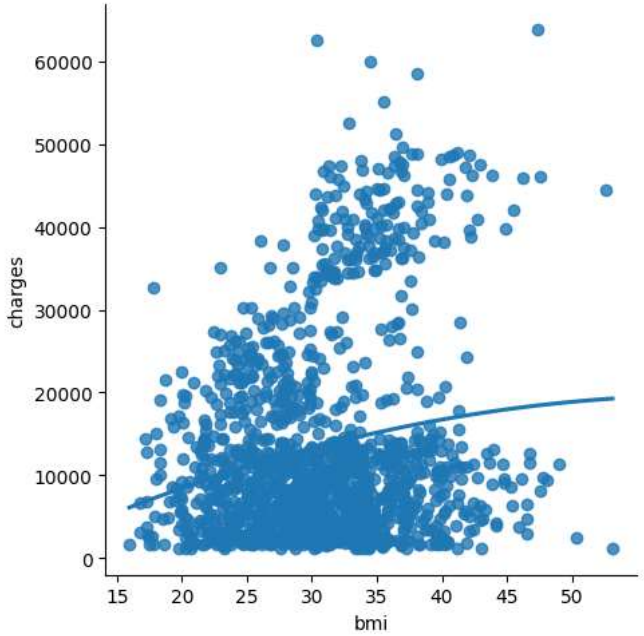


In [11]:
```
plt.scatter(df['age'],df['charges'])
plt.xlabel('age')
plt.ylabel('charges')
```

Out[11]: Text(0, 0.5, 'charges')



In [12]:
```
sns.lmplot(x='bmi',y='charges',order=2,data=df,ci=None)
plt.show()
```

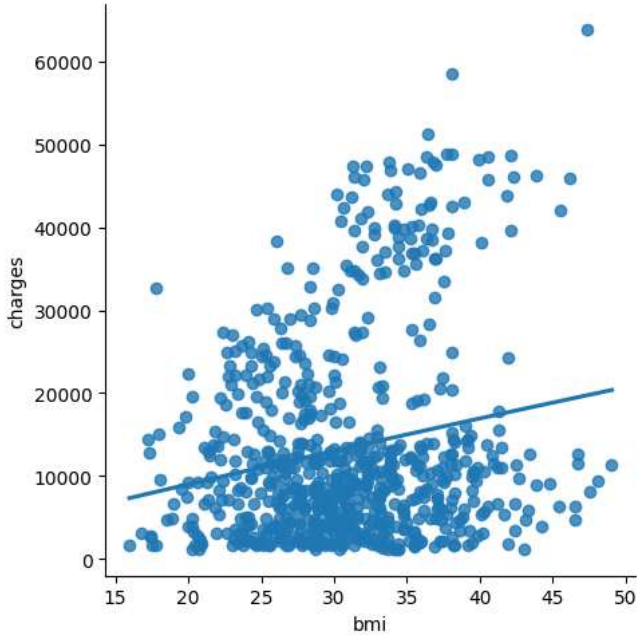


In [13]:
```
x=np.array(df['bmi']).reshape(-1,1)
y=x=np.array(df['charges']).reshape(-1,1)
```

In [25]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
lr=LinearRegression()
lr.fit(x_train,y_train)
print(lr.score(x_test,y_test))
```

```
0.0006810503027022685
```

In [26]:
```
df700=df[:][:700]
sns.lmplot(x='bmi',y='charges',order=2,ci=None,data=df700)
plt.show()
```



In [27]:
```
df700.fillna(method='ffill',inplace=True)
```

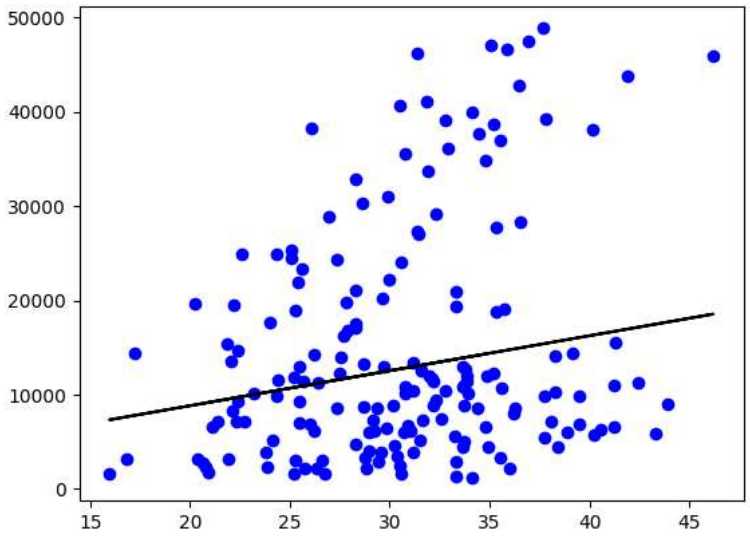
In [28]:
```
x=np.array(df700["bmi"]).reshape(-1,1)
y=np.array(df700['charges']).reshape(-1,1)
```

In [29]:
```
df700.dropna(inplace=True)
```

In [30]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
lr=LinearRegression()
lr.fit(x_train,y_train)
print(lr.score(x_test,y_test))
```

```
0.02485697088491079
```

In [31]:
```
y_pred=lr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test,y_pred,color='k')
plt.show()
```



Evaluation of model

In [32]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [33]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
r2=r2_score(y_test,y_pred)
print(r2)
```

```
0.02485697088491079
```

Ridge Regression

In [34]:
```
from sklearn.linear_model import Lasso,Ridge
from sklearn.preprocessing import StandardScaler
```

In [36]:
```
features=df.columns[0:1]
target=df.columns[-1]
```

In [37]:
```
x=df[features].values
y=df[target].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=1)
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X_test is {}".format(x_test.shape))
```

```
The dimension of X_train is (936, 1)
The dimension of X_test is (402, 1)
```

In [38]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
actual = y_test
train_score_lr = lr.score(x_train, y_train)
test_score_lr = lr.score(x_test, y_test)
print("\nLinear Regression Model:\n")
print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

```
Linear Regression Model:

The train score for lr model is 0.0910963973805714
The test score for lr model is 0.08490473916580776
```

In [39]:
```
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
train_score_ridge = ridgeReg.score(x_train, y_train)
test_score_ridge = ridgeReg.score(x_test, y_test)
print("\nRidge Model:\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(train_score_ridge))
```

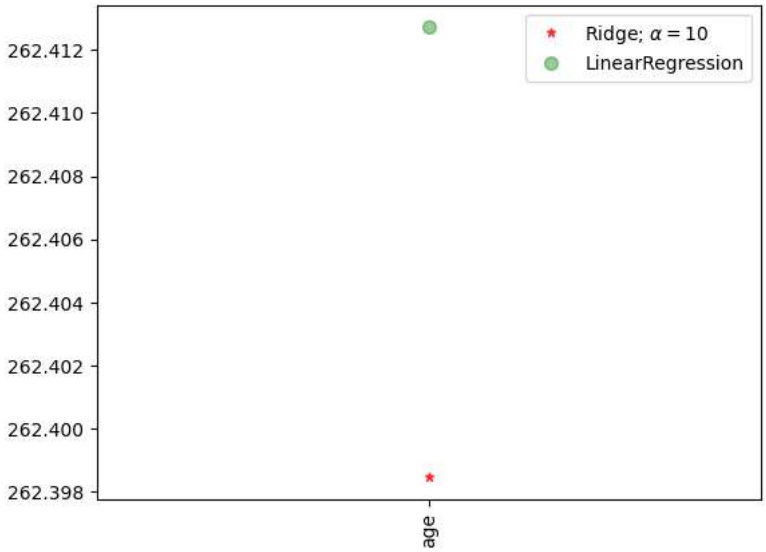
```
Ridge Model:

The train score for ridge model is 0.09109639711159634
The test score for ridge model is 0.09109639711159634
```

In [40]:
```
plt.figure(figsize=(10,10))
```

Out[40]:
```
<Figure size 1000x1000 with 0 Axes>
```

```
<Figure size 1000x1000 with 0 Axes>
```

In [41]:
```
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker="*",markersize=5,color='red',label=r'Ridge; $\alpha=10$',
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker="o",markersize=7,color='green',label='LinearRegression')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Lasso Regression

In [42]:
```
lasso= Lasso(alpha=10)
lasso.fit(x_train,y_train)
train_score_ls = lasso.score(x_train, y_train)
test_score_ls= lasso.score(x_test, y_test)
print("\nRidge Model:\n")
print("The train score for lasso model is {}".format(train_score_ls))
print("The test score for lasso model is {}".format(test_score_ls))
```

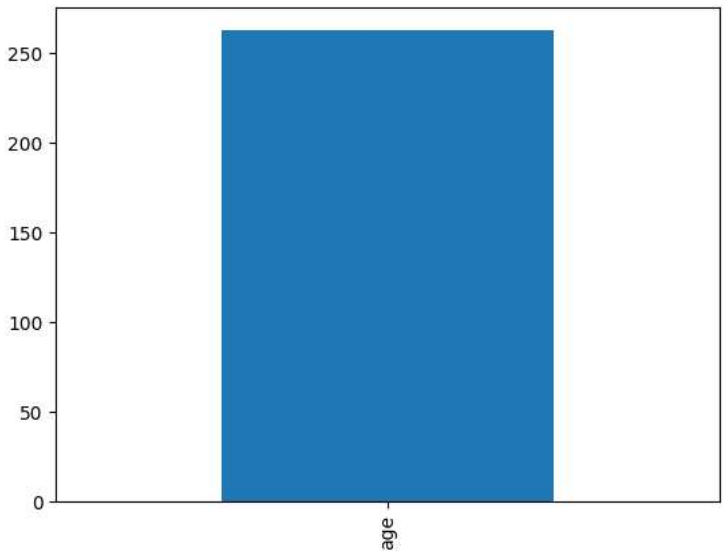
```
Ridge Model:

The train score for lasso model is 0.09109639395809055
The test score for lasso model is 0.08490704421828055
```

In [43]:
```
plt.figure(figsize=(10,10))
```

Out[43]:
```
<Figure size 1000x1000 with 0 Axes>
```


```
<Figure size 1000x1000 with 0 Axes>
```

In [44]:
```
pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "bar")
plt.show()
```



In [45]:
```
from sklearn.linear_model import LassoCV
```

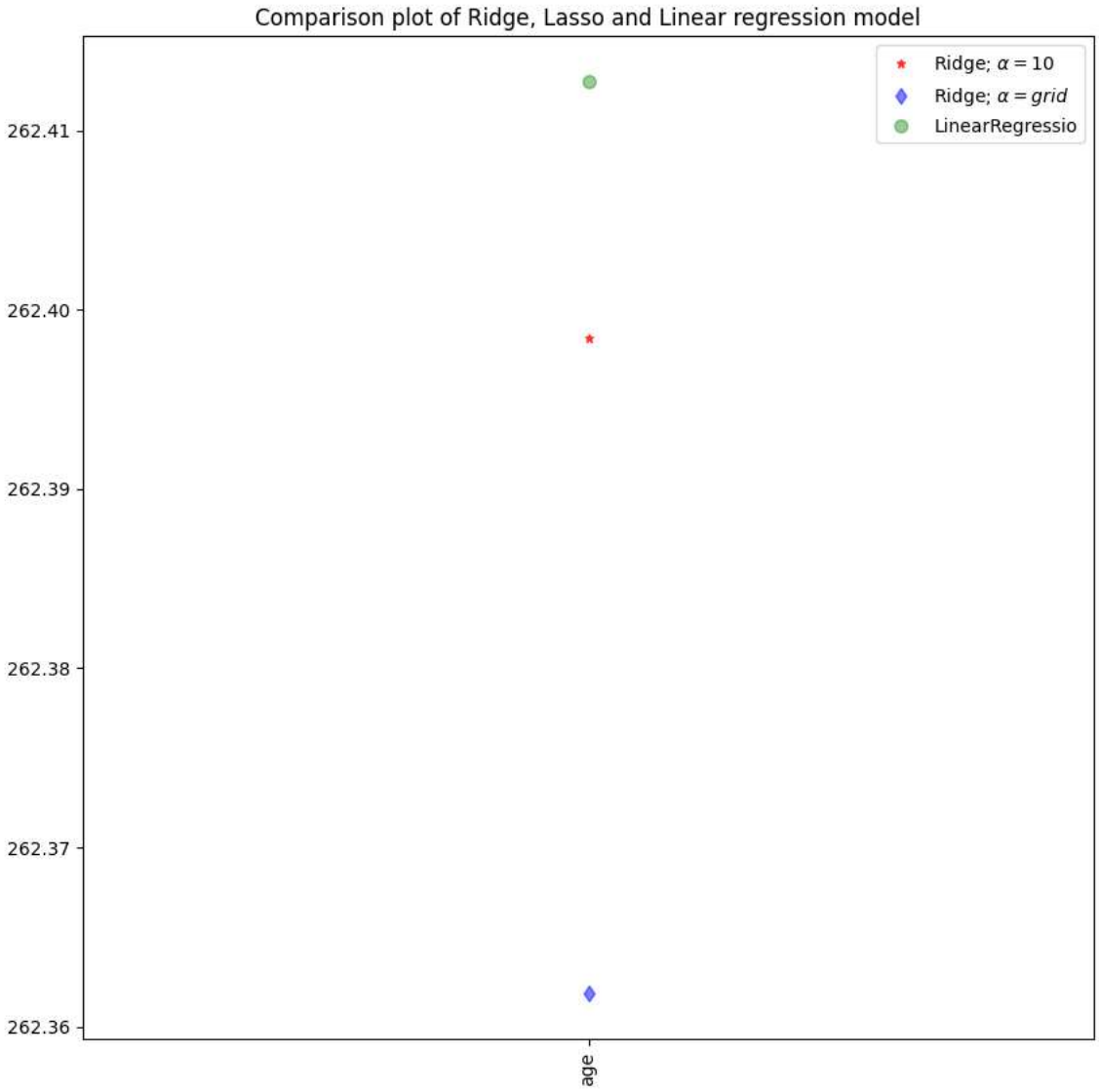
In [48]:
```
from sklearn.linear_model import RidgeCV
ridge_cv=RidgeCV(alphas =[0.0001,0.001,0.01,0.1,1,10]).fit(x_train,y_train)
print(ridge_cv.score(x_train,y_train))
print(ridge_cv.score(x_test,y_test))
```

```
0.09109639711159612
0.08490538609884779
```

In [50]:
```
from sklearn.linear_model import LassoCV
lasso_cv=LassoCV(alphas =[0.0001,0.001,0.01,0.1,1,10]).fit(x_train,y_train)
print(lasso_cv.score(x_train,y_train))
print(lasso_cv.score(x_test,y_test))
```

```
0.09109639395809055
0.08490704421828055
```

In [54]:
```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r'Ridge; $\alpha=10$',
plt.plot(lasso_cv.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Ridge; $\alpha= grid$')
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label='LinearRegressio')
plt.xticks(rotation = 90)
plt.legend()
plt.title("Comparison plot of Ridge, Lasso and Linear regression model")
plt.show()
```



Elastic Regression

In [55]:
```
from sklearn.linear_model import ElasticNet
```

In [56]:
```
el=ElasticNet()
el.fit(x_train,y_train)
print(el.coef_)
print(el.intercept_)
```

```
[261.74450967]
3115.083177426244
```

In [57]:
```
y_pred_elastic=el.predict(x_train)
```

In [58]:
```
mean_squared_error=np.mean((y_pred_elastic-y_train)**2)
print(mean_squared_error)
```

```
135077142.70714515
```

In [59]:
```
el=ElasticNet()
el.fit(x_train,y_train)
print(el.score(x_train,y_train))
```

```
0.09109580670592365
```

Logistic Regression

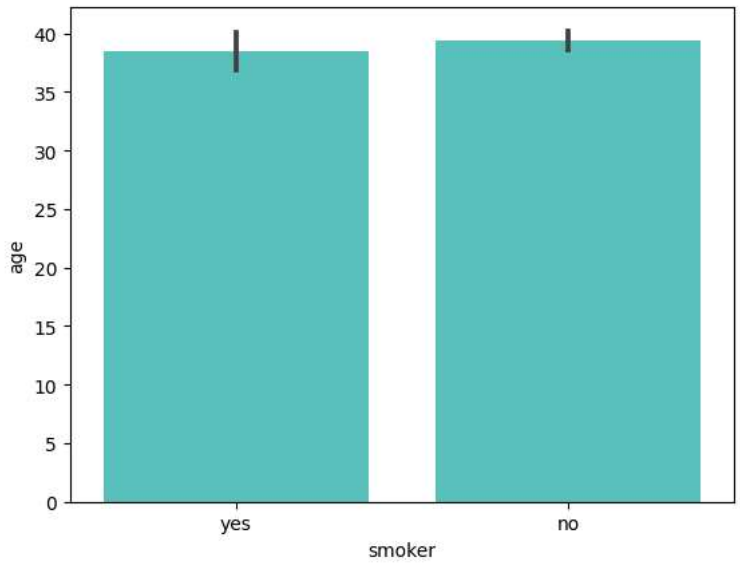
In [60]:
```
features_matrix=df.iloc[:,0:4]
```

In [63]:
```
target_vector=df.iloc[:,-3]
```

In [64]:
```
print('The Feature Matrix has %d Rows and %d columns(s)'%(features_matrix.shape))
print('The Target Matrix has %d Rows and %d columns(s)'%(np.array(target_vector).reshape(-1,1).shape))
```

```
The Feature Matrix has 1338 Rows and 4 columns(s)
The Target Matrix has 1338 Rows and 1 columns(s)
```

In [65]:
```
sns.barplot(x='smoker', y='age', data=df, color="mediumturquoise")
plt.show()
```



In [66]:
```
features_matrix_standardized=StandardScaler().fit_transform(features_matrix)
```

In [67]:
```
algorithm=LogisticRegression(max_iter=10000)
```

In [68]:
```
Logistic_Regression_Model=algorithm.fit(features_matrix_standardized,target_vector)
```

In [69]:
```
observation=[[1,0,0.99539,-0.0588]]
```

In [70]:
```
predictions=Logistic_Regression_Model.predict(observation)
print('The model predicted the observation to belong to class %s'%(predictions))
```

```
The model predicted the observation to belong to class ['no']
```

In [71]:
```
print('The algoritham was trained to predict one of the two classes:%s'%(algorithm.classes_))
```

```
The algoritham was trained to predict one of the two classes:['no' 'yes']
```

In [75]:
```
print("the model says the probability of the observation we passed belonging to class[0] Is %s" " "%(algorithm.predict_proba
print("the model says the probability of the observation we passed belonging to class['1'] Is %s" " "%(algorithm.predict_prol
```

```
the model says the probability of the observation we passed belonging to class[0] Is 0.8057075871331396
the model says the probability of the observation we passed belonging to class['1'] Is 0.8057075871331396
```

In [76]:
```
x=np.array(df['age']).reshape(-1,1)
y=np.array(df['smoker']).reshape(-1,1)
```

In [77]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.05)
lo=LogisticRegression()
lo.fit(x_train,y_train)
print(lo.score(x_test,y_test))
```

```
0.8208955223880597

C:\Users\teppa\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Decision Tree

In [79]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [81]:
```
x=["bmi","children"]
y=["Yes","No"]
all_inputs=df[x]
all_classes=df["sex"]
```

In [82]:
```
(x_train,x_test,y_train,y_test)=train_test_split(all_inputs,all_classes,test_size=0.03)
```

In [83]:
```
clf=DecisionTreeClassifier(random_state=0)
```

In [84]:
```
clf.fit(x_train,y_train)
```

Out[84]:
```
DecisionTreeClassifier(random_state=0)
```

In [85]:
```
score=clf.score(x_test,y_test)
print(score)
```

```
0.4634146341463415
```

Random Forest

In [86]:
```
df['region'].value_counts()
```

Out[86]:
```
region
2    364
1    325
4    325
3    324
Name: count, dtype: int64
```

In [87]:
```
df['bmi'].value_counts()

```

Out[87]:
```
bmi
32.300    13
28.310     9
30.495     8
30.875     8
31.350     8
          ..
46.200     1
23.800     1
44.770     1
32.120     1
30.970     1
Name: count, Length: 548, dtype: int64
```

In [88]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[88]:
```
RandomForestClassifier()
```

In [89]:
```
rf=RandomForestClassifier()
params={'max_depth':[2,3,5,20],
'min_samples_leaf':[5,10,20,50,100,200],
'n_estimators':[10,25,30,50,100,200]}
```

In [90]:
```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rf,param_grid=params,cv=2,scoring="accuracy")
grid_search.fit(x_train,y_train)
```

Out[90]:

GridSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

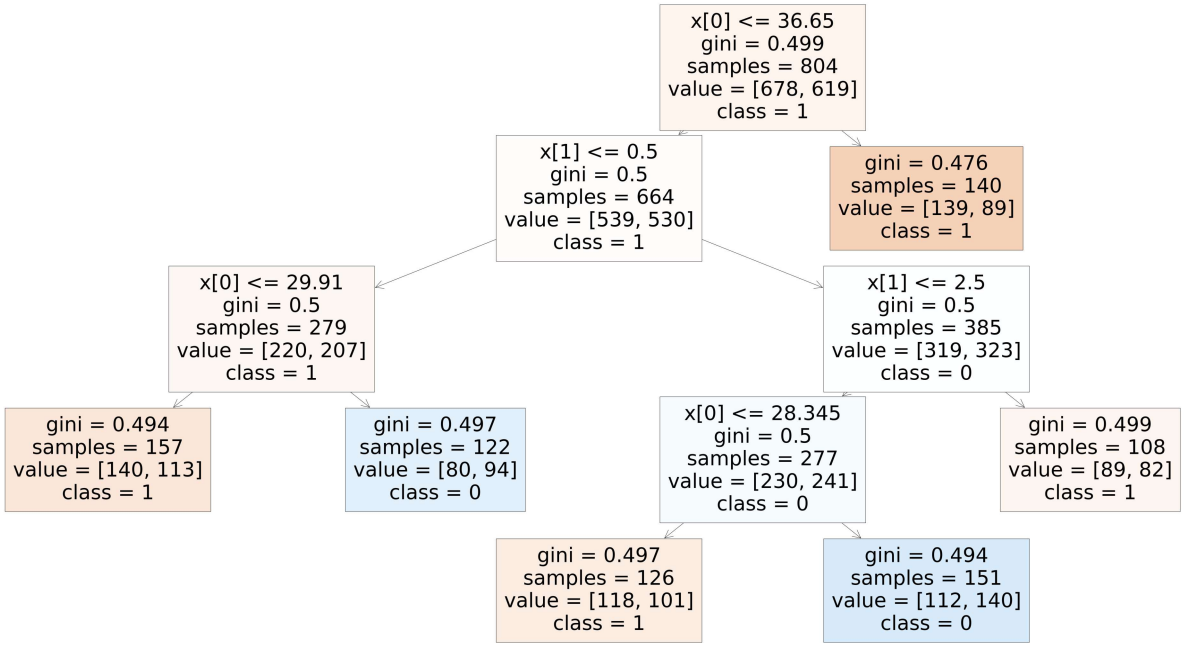
In [91]:
```
grid_search.best_score_
```

Out[91]:
```
0.5289274572466662
```

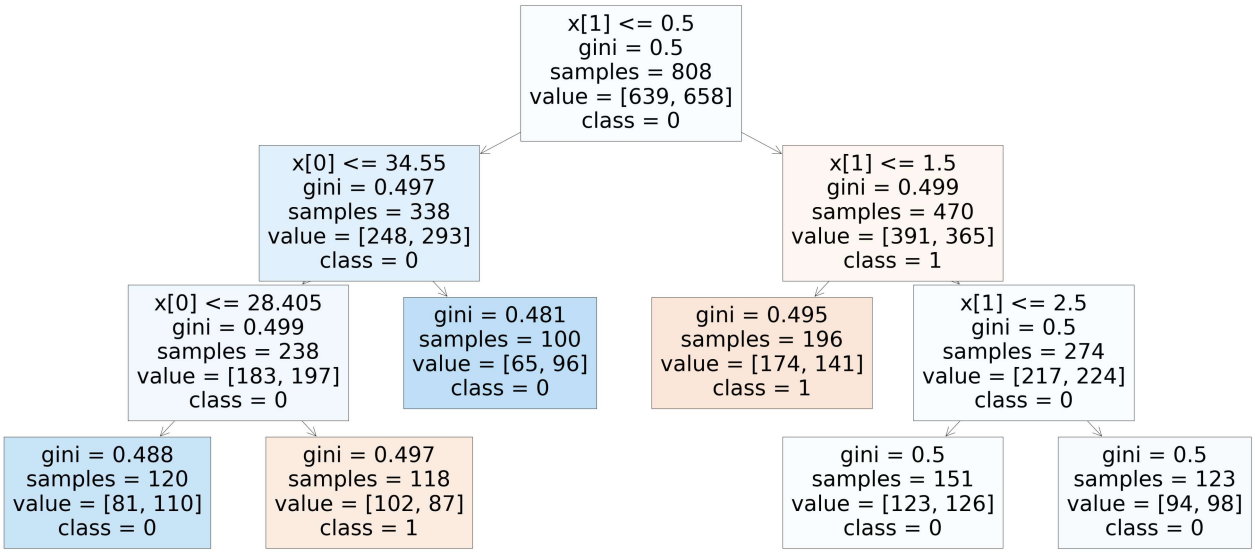
In [92]:
```
rf_best=grid_search.best_estimator_
print(rf_best)
```

```
RandomForestClassifier(max_depth=5, min_samples_leaf=100, n_estimators=25)
```

In [93]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[4],class_names=['1','0'],filled=True);
```



In [94]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(70,30))
plot_tree(rf_best.estimators_[6],class_names=["1","0"],filled=True);

```



In [95]:
```
rf_best.feature_importances_
```

Out[95]:
```
array([0.83811813, 0.16188187])
```

In [96]:
```
rf=RandomForestClassifier(random_state=0)
```

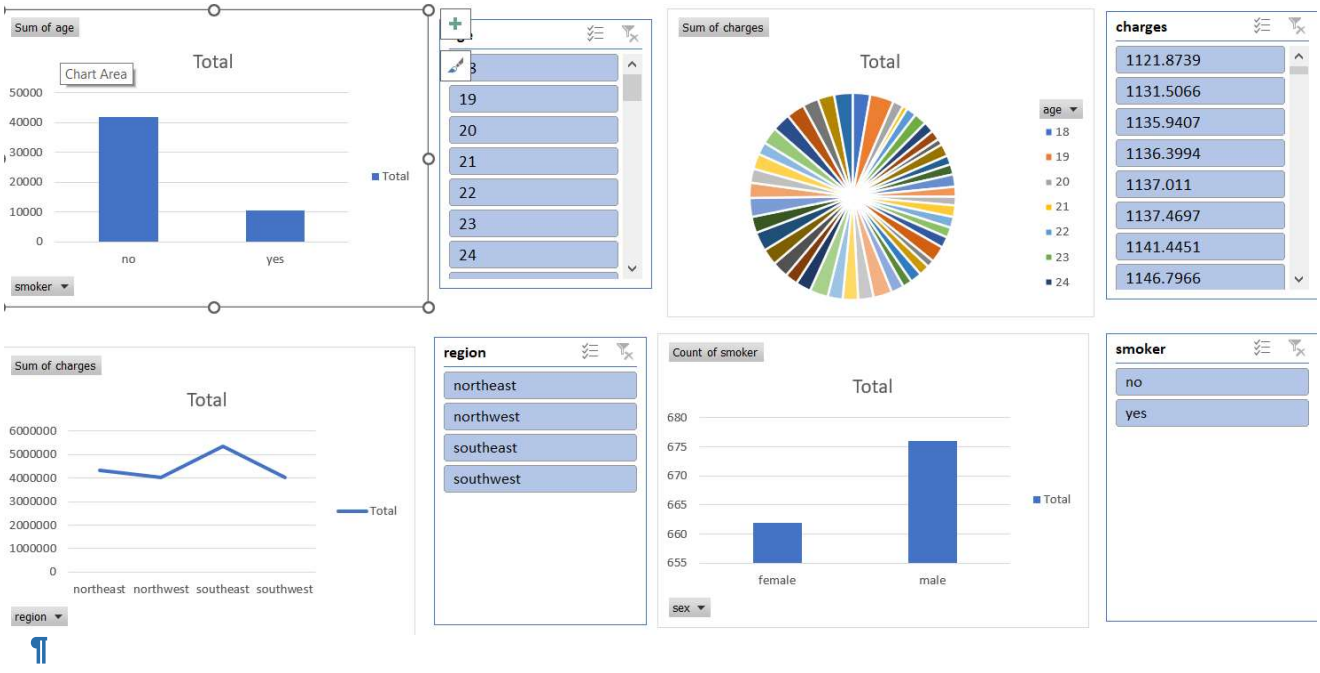
In [97]:
```
rf.fit(x_train,y_train)
```

Out[97]:
```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

In [98]:
```
score=rf.score(x_test,y_test)
print(score)
```

```
0.36585365853658536
```

Data visulation -Excel Data dashboard



¶

conclusion

In [ ]: he model to the given dataset "insurance" Logistic Regression has hightest accuracy 80% hence Logistic Regression is the best fit