PROJECT : HEALTH INSURANCE PREDICTION

- DATE : 19.03.2024
- NAME : yashwanth
- DOMAIN : MACHINE LEARNIG
    - DEGREE : BTECH AI AND DS (3RD YEAR)

DATA SET

```
#importing libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as py
import seaborn as sb
```

```
#reading the dataset (DS)

med_ins=pd.read_csv("insurance.csv")
print(med_ins)
```

```
      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 x 7 columns]
```

```
print(med_ins.nunique())
```

```
age           47
sex            2
bmi          548
children       6
smoker         2
region         4
charges     1337
dtype: int64
```

```
print(med_ins["region"].unique())
```

```
['southwest' 'southeast' 'northwest' 'northeast']
```

```
#information of DS

print(med_ins.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
None
```

```
#checking null values in DS

print(med_ins.isna().sum())
```

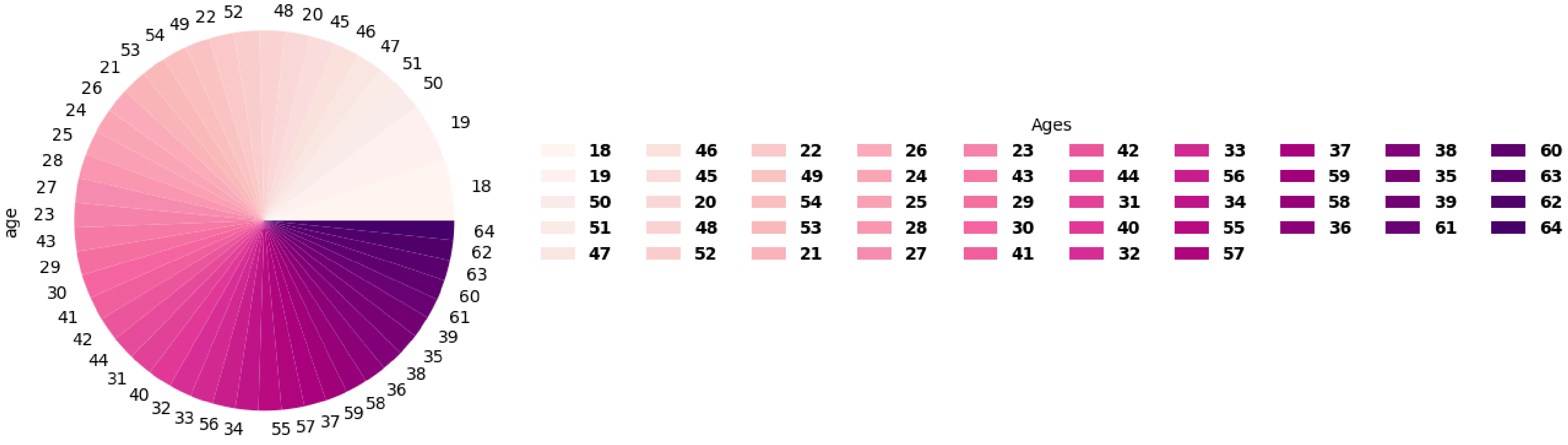
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

```
#displaying the columns

print(med_ins.columns)
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

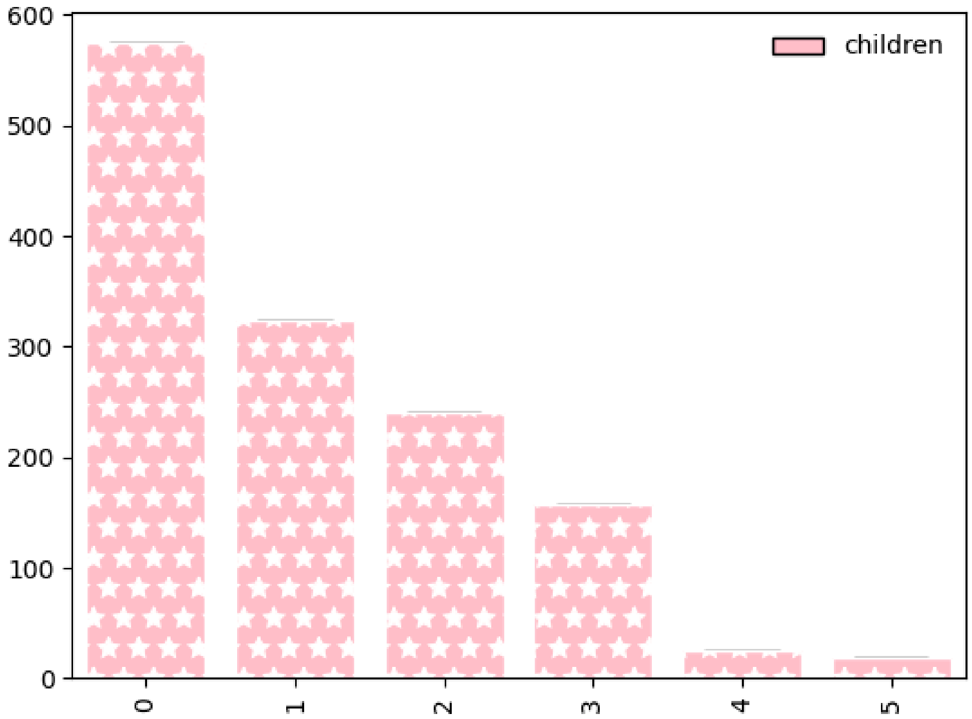
```
py.figure(figsize=(5,8))
med_ins['age'].value_counts().plot(kind='pie',colormap='RdPu')
py.legend(title='Ages',bbox_to_anchor=(1.05,0.75),ncol=10,loc='best',frameon=False,prop ={'weight':'bold'},fontsize=6)
```

```
<matplotlib.legend.Legend at 0x7ddf06236860>
```



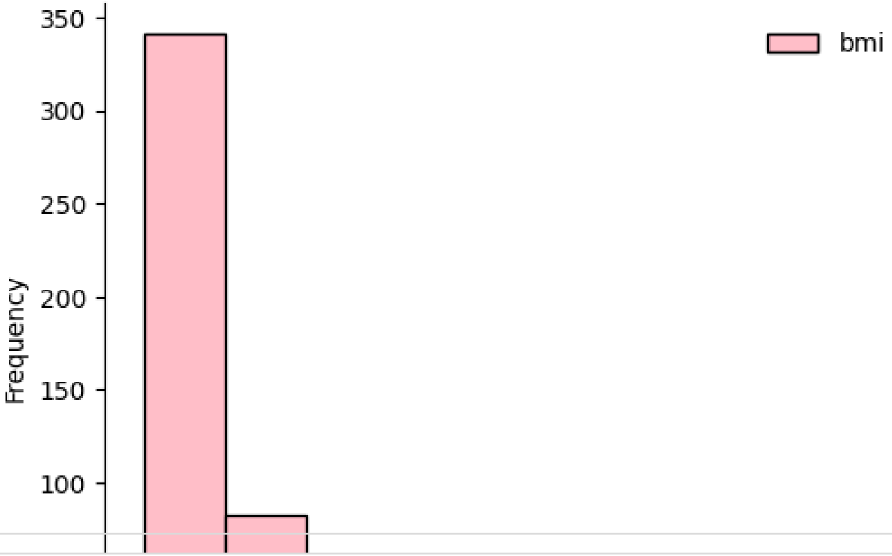
```
a=med_ins['children'].value_counts()
a.plot(kind='bar',color='pink',edgecolor='black')
py.legend(loc='best',frameon=False)
py.bar(range(len(a)),a,hatch='*',color='pink',edgecolor='white')
```

```
<BarContainer object of 6 artists>
```

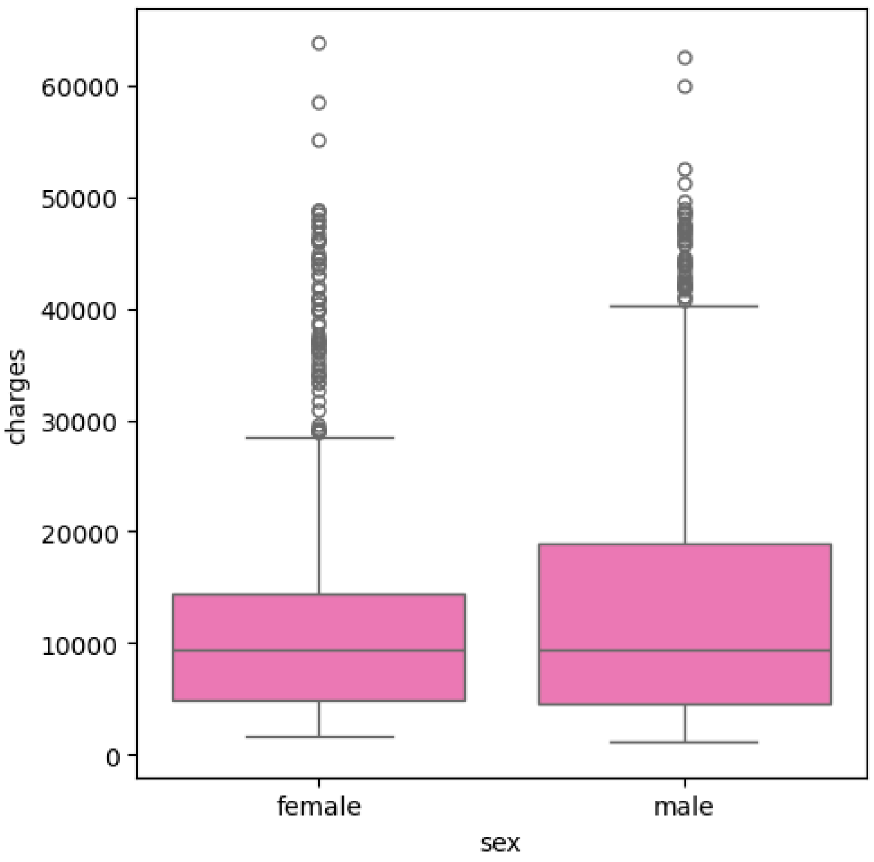


```
b=med_ins['bmi'].value_counts().plot(kind='hist',color='pink',edgecolor='black')
ax1=py.gca()
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
py.legend(loc='best',frameon=False,bbox_to_anchor=(0.9,0.99))
```

```
<matplotlib.legend.Legend at 0x7ddf06235ff0>
```

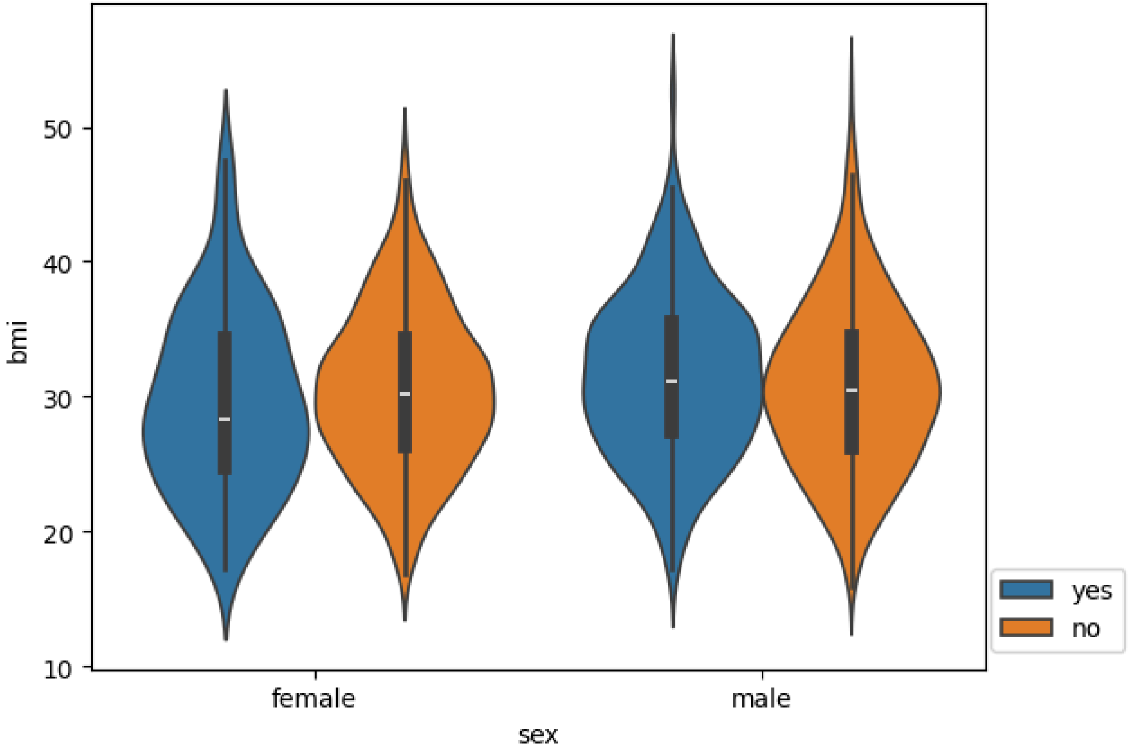


```
sb.catplot(x="sex",y="charges",kind='box',color='hotpink',data=med_ins)
#ax=py.axes()
#ax.set_facecolor("black")
ax1=py.gca()
ax1.spines['top'].set_visible(True)
ax1.spines['right'].set_visible(True)
```



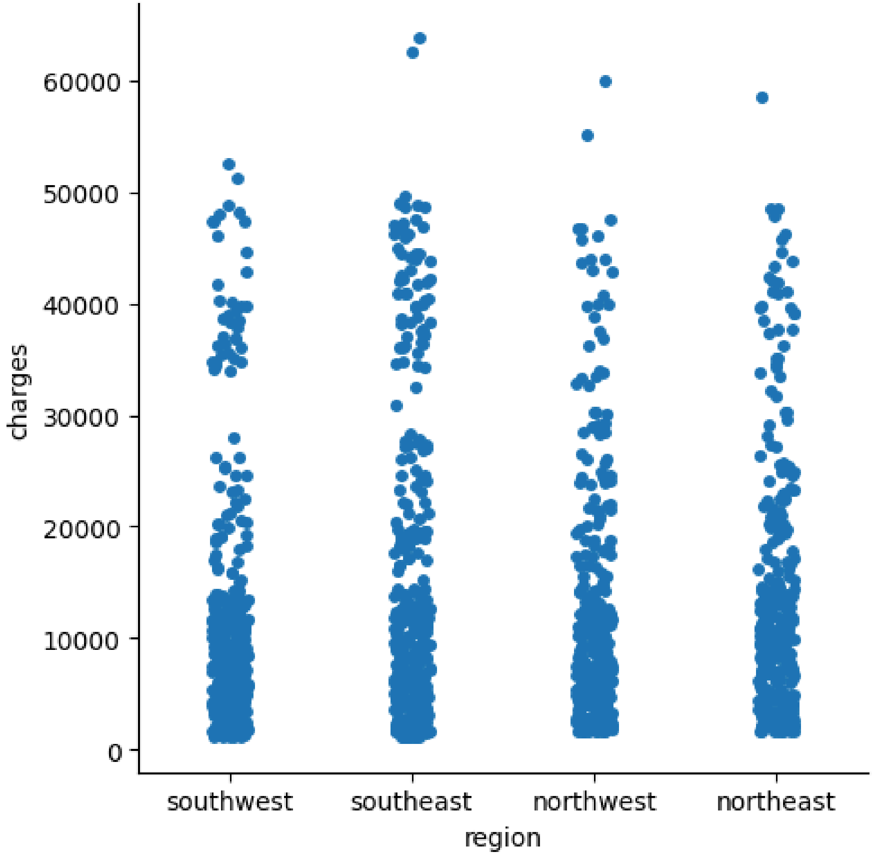
```
sb.violinplot(x="sex",y="bmi",hue="smoker",data=med_ins)
py.legend(bbox_to_anchor=(1.17,0.17),frameon=True)
```

```
<matplotlib.legend.Legend at 0x7ddf03fdf010>
```



```
sb.catplot(x="region",y="charges",data=med_ins)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ddf0236a170>
```



```
#decription of DS

print(med_ins.describe())
```

```
                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
```

```
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
med_ins.iloc[:,1]=l.fit_transform(med_ins.iloc[:,1])

med_ins.iloc[:,4]=l.fit_transform(med_ins.iloc[:,4])

med_ins.iloc[:,5]=l.fit_transform(med_ins.iloc[:,5])

print(med_ins)
```

```
      age  sex     bmi  children  smoker  region      charges
0      19    0  27.900         0       1       3  16884.92400
1      18    1  33.770         1       0       2   1725.55230
2      28    1  33.000         3       0       2   4449.46200
3      33    1  22.705         0       0       1  21984.47061
4      32    1  28.880         0       0       1   3866.85520
...   ...  ...     ...       ...     ...     ...          ...
1333   50    1  30.970         3       0       1  10600.54830
1334   18    0  31.920         0       0       0   2205.98080
1335   18    0  36.850         0       0       2   1629.83350
1336   21    0  25.800         0       0       3   2007.94500
1337   61    0  29.070         0       1       1  29141.36030

[1338 rows x 7 columns]
<ipython-input-15-3b6b98c751e9>:3: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpla
  med_ins.iloc[:,1]=l.fit_transform(med_ins.iloc[:,1])
<ipython-input-15-3b6b98c751e9>:5: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpla
  med_ins.iloc[:,4]=l.fit_transform(med_ins.iloc[:,4])
<ipython-input-15-3b6b98c751e9>:7: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpla
  med_ins.iloc[:,5]=l.fit_transform(med_ins.iloc[:,5])
```

LinearRegression

```
#assign x and y

x=med_ins[['smoker']].values
y=med_ins[['charges']].values
```

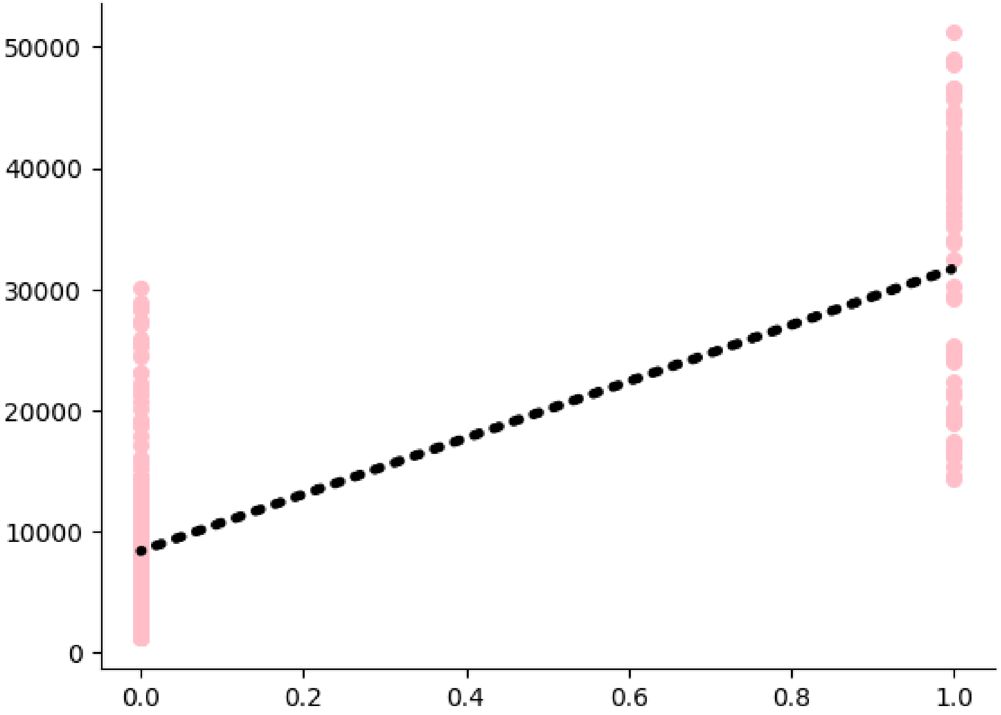
```
#spliting

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
#algorithm

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr1=lr.fit(x_train,y_train)
print('COEFFICIENT:',lr1.coef_)
print('  INTERCEPT:',lr1.intercept_)
```

```
COEFFICIENT: [[23332.42129457]]
  INTERCEPT: [8354.09098751]
```

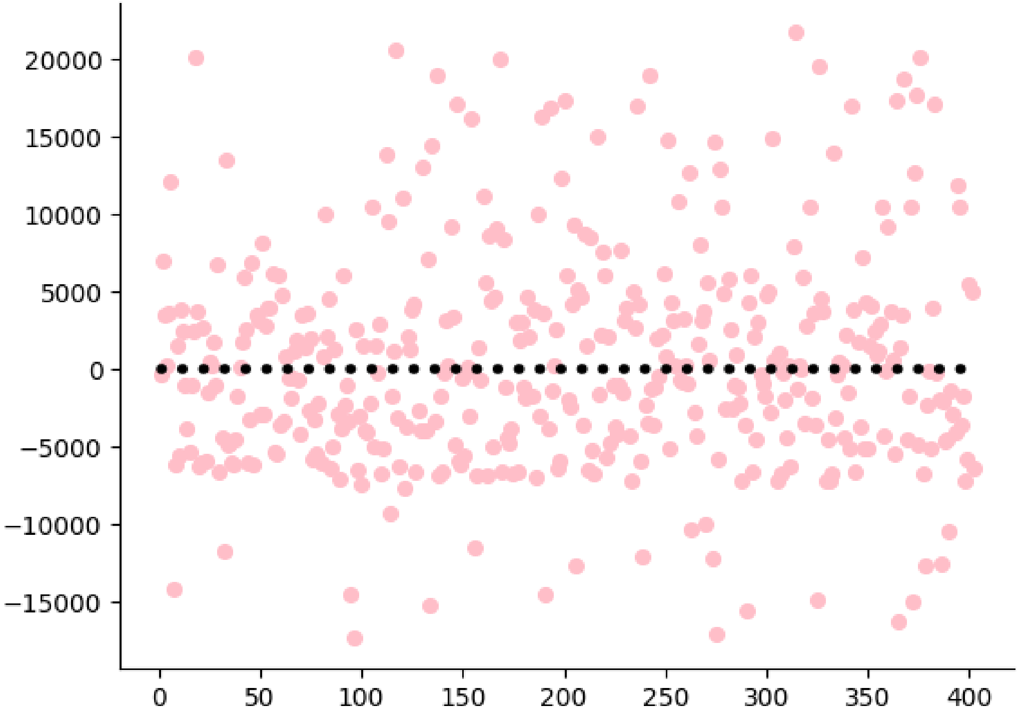
```
#prediction

y_pred=lr1.predict(x_test)
ax=py.axes()
ax1=py.gca()
ax.set_facecolor("white")
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
py.scatter(x_test,y_test,s=30,color="pink")
py.plot(x_test,y_pred,color="black",linewidth=4,linestyle=(0,(0.1,2)),dash_capstyle="round")
```

```
[<matplotlib.lines.Line2D at 0x7ddefefd1f30>]
```



```
#Difference prediction

res=y_test-y_pred
y=[i for i in range(1,len(res)+1)]
ax=py.axes()
ax1=py.gca()
ax.set_facecolor("white")
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
py.scatter(y,res,s=30,color='pink')
py.plot(y,[0]*len(y_pred),color='black',linewidth=4,linestyle=(0,(0.1,2)),dash_capstyle="round")
```

```
[<matplotlib.lines.Line2D at 0x7ddeff047d60>]
```



```
from sklearn.metrics import r2_score,mean_squared_error
mse = mean_squared_error(y_test,y_pred)
print("ROOT MEAN SQUARED ERROR : ",np.sqrt(mse))
print("     MEAN SQUARED ERROR : ",mse)
A1_=r2_score(y_test,y_pred)
print("         ACCUARCY SCORE :",A1_)
```

```
ROOT MEAN SQUARED ERROR :  7165.756777678182
     MEAN SQUARED ERROR :  51348070.19684079
         ACCUARCY SCORE : 0.6406587599098937
```

PRE PROCESSING

```
#assign x and y
```

```
x=med_ins.iloc[:,0:-1].values
y=med_ins.iloc[:,-1].values
```

```
#split

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
#Normalize

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

DECISION TREE REGRESSOR

```
#Algorithm

from sklearn.tree import DecisionTreeRegressor
dtr  = DecisionTreeRegressor()
dtr1 = dtr.fit(x_train,y_train)
```

```
#Predict
y_pred = dtr1.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error
a1=mean_squared_error(y_test,y_pred)
r1=print('ROOT MEAN SQUARED ERROR : ',np.sqrt(a1))
print("     MEAN SQUARED ERROR : ",a1)
B1_=r2_score(y_test,y_pred)
print('         ACCUARCY SCORE :',B1_)
```

```
ROOT MEAN SQUARED ERROR :  6680.25433580387
     MEAN SQUARED ERROR :  44625797.99102641
         ACCUARCY SCORE : 0.6811991216237633
```

NAIVE BAYES

```
#Algorithm

from sklearn.linear_model import BayesianRidge
BR=BayesianRidge()
BR2=BR.fit(x_train,y_train)
```

```
#predict

y_pred=BR2.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error
a1=mean_squared_error(y_test,y_pred)
r1=print('ROOT MEAN SQUARED ERROR : ',np.sqrt(a1))
print("     MEAN SQUARED ERROR : ",a1)
C1_=r2_score(y_test,y_pred)
print('         ACCUARCY SCORE :',C1_)
```

```
ROOT MEAN SQUARED ERROR :  5976.886019951411
     MEAN SQUARED ERROR :  35723166.49549062
         ACCUARCY SCORE : 0.7447983594728539
```

RANDOM FOREST TREE

```
#ALGORITHM

from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf1=rf.fit(x_train,y_train)
```

```
#predict

y_pred=rf1.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error
a1=mean_squared_error(y_test,y_pred)
r1=print('ROOT MEAN SQUARED ERROR : ',np.sqrt(a1))
print("     MEAN SQUARED ERROR : ",a1)
D1_=r2_score(y_test,y_pred)
print('         ACCUARCY SCORE :',D1_)
```

```
ROOT MEAN SQUARED ERROR :  4707.5588644641
     MEAN SQUARED ERROR :  22161110.462394528
         ACCUARCY SCORE : 0.8416839182881424
```

K-NEAREST NEIGHBOUR

```
#ALGORITHM

from sklearn.neighbors import KNeighborsRegressor
kn=KNeighborsRegressor(n_neighbors=5,metric='minkowski',p=2)
kn1=kn.fit(x_train,y_train)
```

```
#predict

y_pred=kn1.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error
a1=mean_squared_error(y_test,y_pred)
r1=print('ROOT MEAN SQUARED ERROR : ',np.sqrt(a1))
print("     MEAN SQUARED ERROR : ",a1)
E1_=r2_score(y_test,y_pred)
print('         ACCUARCY SCORE :',E1_)
```

```
ROOT MEAN SQUARED ERROR :  5092.860629799064
     MEAN SQUARED ERROR :  25937229.394557316
         ACCUARCY SCORE : 0.8147078173191759
```

SUPPORT VECTOR MACHINE (SVM)

```
#ALGORITHM

from sklearn.svm import SVR
sv=SVR()
sv1=sv.fit(x_train,y_train)
```

```
#predict
y_pred=sv1.predict(x_test)

from sklearn.metrics import r2_score,mean_squared_error
a1=mean_squared_error(y_test,y_pred)
r1=print('ROOT MEAN SQUARED ERROR : ',np.sqrt(a1))
print("     MEAN SQUARED ERROR : ",a1)
F1_=r2_score(y_test,y_pred)
print('         ACCUARCY SCORE :',F1_)
```

```
ROOT MEAN SQUARED ERROR :  12719.78303542988
     MEAN SQUARED ERROR :  161792880.46840978
         ACCUARCY SCORE : -0.15582722842788033
```

FEATURE IMPORTANCE

```
lst=[A1_,B1_,C1_,D1_,E1_,F1_]
print(       )
print('                                   ALGORITHM ACCURACY SCORES       ')
print(     )
print(     )
print("LINEAR REGRESSION ACCUARY SCORE : ",A1_)
print("RANDOM FOREST ACCURACY SCORE     : ",D1_)
print("DECISION TREE ACCURACY SCORE     : ",B1_)
print("NAIVE BAYES ACCURACY SCORE       : ",C1_)
print("KNN ACCURACY SCORE               : ",E1_)
print("SVM ACCURACY SCORE               :",F1_)
print(       )
print(       )
print(       )

max_=max(lst)

if max_==A1_:

    print('SELECTED ALGORITHM : LINEAR ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCURACY SCORE    :',max_)
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance=lr.feature_importances_
    for i,v in enumerate(importance):
        print("Feature: %0d ; Score: %5f"%(i,v))

    index=med_ins.columns[:-1]
    importance=pd.Series(rf.feature_importances_,index=index)
    importance.nlargest(13).plot(kind='barh',color='pink')

elif max_==B1_:

    print('SELECTED ALGORITHM : DECISION TREE: ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCURACY SCORE    :',max_)
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance1=dtr.feature_importances_
    for i,v in enumerate(importance1):
        print("Feature: %0d ; Score: %5f"%(i,v))
    index1=med_ins.columns[:-1]
    importance1=pd.Series(rf.feature_importances_,index=index1)
    importance1.nlargest(13).plot(kind='barh',color='pink')

elif max_==C1_:
    print('SELECTED ALGORITHM : NAIVE BAYES: ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCURACY SCORE    :',max_)
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance2=BR.feature_importances_
    for i,v in enumerate(importance2):
        print("Feature: %0d ; Score: %5f"%(i,v))

    index2=med_ins.columns[:-1]
    importance2=pd.Series(rf.feature_importances_,index=index2)
    importance2.nlargest(13).plot(kind='barh',color='pink')


elif max_==D1_:

    print('SELECTED ALGORITHM : RANDOM FOREST ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCUARACY SCORE    :',max_)
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance3=rf.feature_importances_
    for i,v in enumerate(importance3):
        print("Feature: %0d ; Score: %5f"%(i,v))

    index3=med_ins.columns[:-1]
    importance3=pd.Series(rf.feature_importances_,index=index3)
    importance3.nlargest(13).plot(kind='barh',color='pink')

elif max_==E1_:

    print('SELECTED ALGORITHM :KNN ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCUARACY SCORE    :',max_)
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance4=kn.feature_importances_
    for i,v in enumerate(importance4):
        print("Feature: %0d ; Score: %5f"%(i,v))

    index4=med_ins.columns[:-1]
    importance4=pd.Series(rf.feature_importances_,index=index4)
    importance4.nlargest(13).plot(kind='barh',color='pink')

elif max_==F1_:
    print('SVM:',max_)
    print('SELECTED ALGORITHM : SVM ')
    print('TYPE OF ALGORITHM  : REGRESSION')
    print('ACCUARACY SCORE    :',max_)
    print(       )
    print(       )
    print('CONTRIBUTION')
    print(       )

    importance5=sv.feature_importances_
    for i,v in enumerate(importance5):
        print("Feature: %0d ; Score: %5f"%(i,v))

    index5=med_ins.columns[:-1]
    importance5=pd.Series(rf.feature_importances_,index=index5)
    importance5.nlargest(13).plot(kind='barh',color='pink')

else:
    print('OOPS!! SOMETHING WENT WRONG')
```

```
                                     ALGORITHM ACCURACY SCORES
```