IMPORTING THE LIBRARIES

```
import pandas as pd
from pandas import read_csv
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,r2_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import warnings
warnings.filterwarnings("ignore")
```

LOADING THE DATASET

```
data=pd.read_csv(r"C:\Users\Admin\Downloads\
1651277648862_healthinsurance.csv")
data.head()
```

```
    age     sex  weight   bmi hereditary_diseases  no_of_dependents
smoker  \
0  60.0    male      64  24.3           NoDisease                 1
0
1  49.0  female      75  22.6           NoDisease                 1
0
2  32.0  female      64  17.8            Epilepsy                 2
1
3  61.0  female      53  36.4           NoDisease                 1
1
4  19.0  female      50  20.6           NoDisease                 0
0

          city  bloodpressure  diabetes  regular_ex    job_title
claim
0      NewYork             72         0           0        Actor
13112.6
1       Boston             78         1           1     Engineer
9567.0
2  Phildelphia             88         1           1   Academician
32734.2
```

```
3    Pittsburg             72         1           0         Chef
48517.6
4      Buffalo             82         1           0   HomeMakers
1731.7

data.shape

(15000, 13)

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  14604 non-null  float64
 1   sex                  15000 non-null  object 
 2   weight               15000 non-null  int64  
 3   bmi                  14044 non-null  float64
 4   hereditary_diseases  15000 non-null  object 
 5   no_of_dependents     15000 non-null  int64  
 6   smoker               15000 non-null  int64  
 7   city                 15000 non-null  object 
 8   bloodpressure        15000 non-null  int64  
 9   diabetes             15000 non-null  int64  
 10  regular_ex           15000 non-null  int64  
 11  job_title            15000 non-null  object 
 12  claim                15000 non-null  float64
dtypes: float64(3), int64(6), object(4)
memory usage: 1.5+ MB

data.isnull().sum()

age                    396
sex                      0
weight                   0
bmi                    956
hereditary_diseases      0
no_of_dependents         0
smoker                   0
city                     0
bloodpressure            0
diabetes                 0
regular_ex               0
job_title                0
claim                    0
dtype: int64

data.dropna()
```

```
        age  sex  weight   bmi  hereditary_diseases  no_of_dependents
smoker  \
0      60.0    1      64  24.3                    8                 1
0
1      49.0    0      75  22.6                    8                 1
0
2      32.0    0      64  17.8                    4                 2
1
3      61.0    0      53  36.4                    8                 1
1
4      19.0    0      50  20.6                    8                 0
0
...     ...  ...     ...   ...                  ...               ...
...
14995  39.0    1      49  28.3                    8                 1
1
14996  39.0    1      74  29.6                    8                 4
0
14997  20.0    1      62  33.3                    8                 0
0
14998  52.0    1      88  36.7                    8                 0
0
14999  52.0    1      57  26.4                    8                 3
0

       city  bloodpressure  diabetes  regular_ex  job_title    claim
0        55             72         0           0          2  13112.6
1         5             78         1           1         16   9567.0
2        63             88         1           1          0  32734.2
3        64             72         1           0         10  48517.6
4         8             82         1           0         22   1731.7
...     ...            ...       ...         ...        ...      ...
14995    24             54         1           0         20  21082.2
14996    49             64         1           0         33   7512.3
14997    82             52         1           0         18   1391.5
14998    61             70         1           0         17   9144.6
14999    37             72         1           0         28  25992.8

[13648 rows x 13 columns]
```

```
data[:]=np.nan_to_num(data)

data.describe()
```

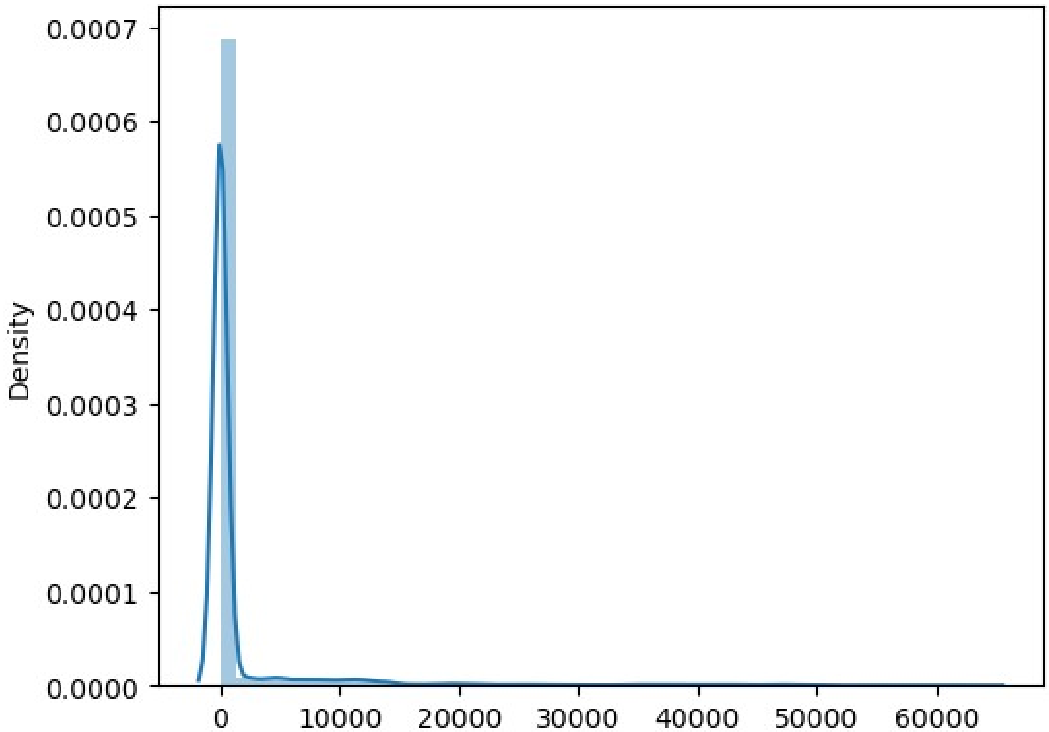
```
                age           sex        weight           bmi  \
count  15000.000000  15000.000000  15000.000000  15000.000000
mean      38.503467      0.489867     64.909600     28.337433
std       15.213913      0.499914     13.701935      9.474541
min        0.000000      0.000000     34.000000      0.000000
25%       26.000000      0.000000     54.000000     25.000000
```

```
50%       40.000000      0.000000     63.000000     28.800000
75%       51.000000      1.000000     76.000000     34.100000
max       64.000000      1.000000     95.000000     53.100000

       hereditary_diseases  no_of_dependents        smoker
city  \
count          15000.000000      15000.000000  15000.000000
15000.000000
mean               7.730533          1.129733      0.198133
45.160000
std                1.251250          1.228469      0.398606
25.930775
min                0.000000          0.000000      0.000000
0.000000
25%                8.000000          0.000000      0.000000
23.000000
50%                8.000000          1.000000      0.000000
47.000000
75%                8.000000          2.000000      0.000000
67.000000
max                9.000000          5.000000      1.000000
90.000000

       bloodpressure      diabetes    regular_ex     job_title
claim
count   15000.000000  15000.000000  15000.000000  15000.000000
15000.000000
mean       68.650133      0.777000      0.224133     18.662267
13401.437620
std        19.418515      0.416272      0.417024     10.429298
12148.239619
min         0.000000      0.000000      0.000000      0.000000
1121.900000
25%        64.000000      1.000000      0.000000     10.000000
4846.900000
50%        71.000000      1.000000      0.000000     20.000000
9545.650000
75%        80.000000      1.000000      0.000000     28.000000
16519.125000
max       122.000000      1.000000      1.000000     34.000000
63770.400000

data.keys()

Index(['age', 'sex', 'weight', 'bmi', 'hereditary_diseases',
       'no_of_dependents', 'smoker', 'city', 'bloodpressure',
'diabetes',
       'regular_ex', 'job_title', 'claim'],
      dtype='object')
```

```
le=LabelEncoder()
data['sex']=le.fit_transform(data['sex'])
data['sex'].unique()

data['hereditary_diseases']=le.fit_transform(data['hereditary_diseases
'])
data['hereditary_diseases'].unique()

data['city']=le.fit_transform(data['city'])
data['city'].unique()

data['job_title']=le.fit_transform(data['job_title'])
data['job_title'].unique()


array([ 2, 16,  0, 10, 22, 12, 32, 13, 30, 33, 15, 28, 29,  5,  8,  6,
9,
       26,  1, 19, 34, 18,  4, 23, 20,  7, 31, 14,  3, 11, 24, 17, 25,
27,
       21], dtype=int64)
```

VISUALIZING THE DATA

```
features =
[['age','sex','weight','bmi','smoker','bloodpressure','diabetes','clai
m']]

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.distplot(data[col])
plt.show()
```



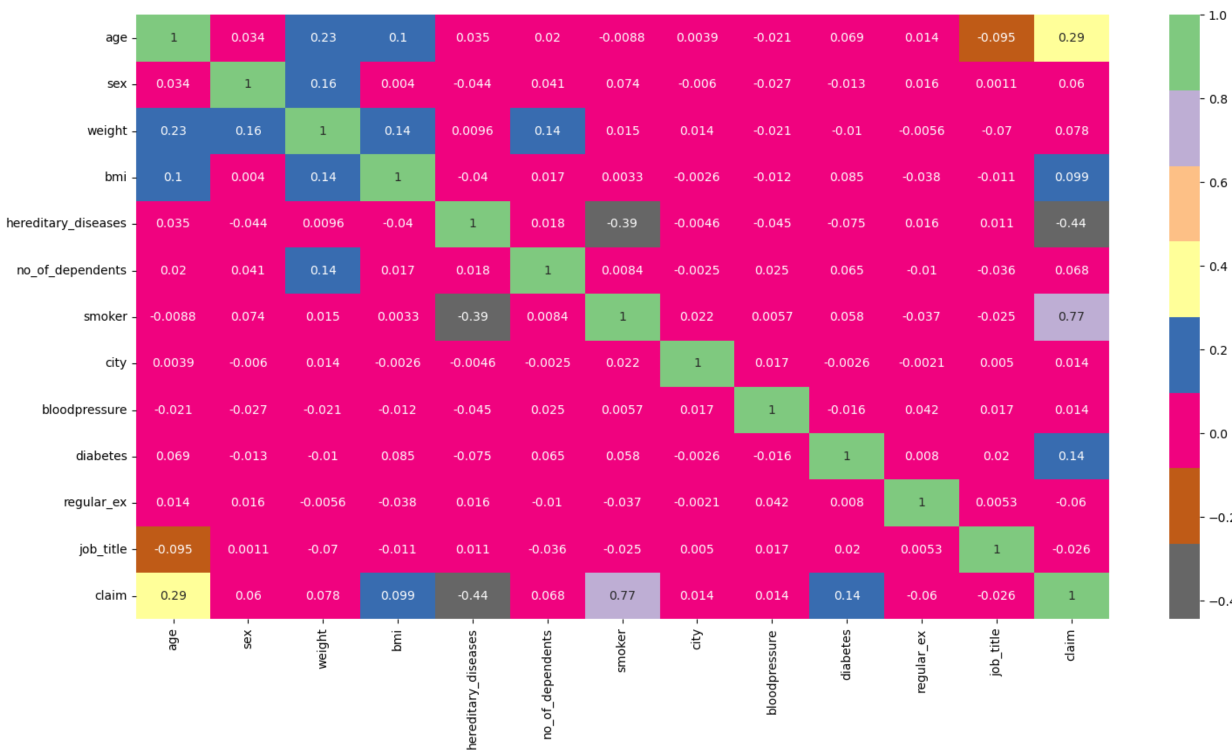

```
data.corr()

                          age       sex    weight       bmi  \
age                  1.000000  0.033912  0.230385  0.101080
sex                  0.033912  1.000000  0.159249  0.003964
weight               0.230385  0.159249  1.000000  0.136462
bmi                  0.101080  0.003964  0.136462  1.000000
hereditary_diseases  0.035083 -0.043909  0.009596 -0.039503
no_of_dependents     0.020042  0.041440  0.135687  0.017216
smoker              -0.008754  0.073981  0.015499  0.003260
city                 0.003866 -0.005995  0.013662 -0.002592
bloodpressure       -0.020699 -0.026718 -0.020835 -0.012007
diabetes             0.068966 -0.012622 -0.010490  0.085160
regular_ex           0.013812  0.016332 -0.005578 -0.038384
job_title           -0.094952  0.001134 -0.070038 -0.010686
claim                0.294430  0.059592  0.077716  0.098840

                     hereditary_diseases  no_of_dependents
smoker  \
age                             0.035083          0.020042 -0.008754

sex                            -0.043909          0.041440  0.073981

weight                          0.009596          0.135687  0.015499

bmi                            -0.039503          0.017216  0.003260
```

```
hereditary_diseases            1.000000          0.018364 -0.390082

no_of_dependents               0.018364          1.000000  0.008364

smoker                        -0.390082          0.008364  1.000000

city                          -0.004599         -0.002519  0.022218

bloodpressure                 -0.045446          0.024849  0.005709

diabetes                      -0.075312          0.065182  0.058164

regular_ex                     0.016476         -0.010302 -0.036949

job_title                      0.010820         -0.035915 -0.025327

claim                         -0.444337          0.067614  0.773399


                         city  bloodpressure  diabetes  regular_ex
job_title  \
age                  0.003866      -0.020699  0.068966    0.013812  -
0.094952
sex                 -0.005995      -0.026718 -0.012622    0.016332
0.001134
weight               0.013662      -0.020835 -0.010490   -0.005578  -
0.070038
bmi                 -0.002592      -0.012007  0.085160   -0.038384  -
0.010686
hereditary_diseases -0.004599      -0.045446 -0.075312    0.016476
0.010820
no_of_dependents    -0.002519       0.024849  0.065182   -0.010302  -
0.035915
smoker               0.022218       0.005709  0.058164   -0.036949  -
0.025327
city                 1.000000       0.016921 -0.002642   -0.002071
0.005045
bloodpressure        0.016921       1.000000 -0.016498    0.042493
0.017182
diabetes            -0.002642      -0.016498  1.000000    0.007960
0.019815
regular_ex          -0.002071       0.042493  0.007960    1.000000
0.005342
job_title            0.005045       0.017182  0.019815    0.005342
1.000000
claim                0.013785       0.013742  0.135371   -0.060492  -
0.026016

                        claim
age                  0.294430
```

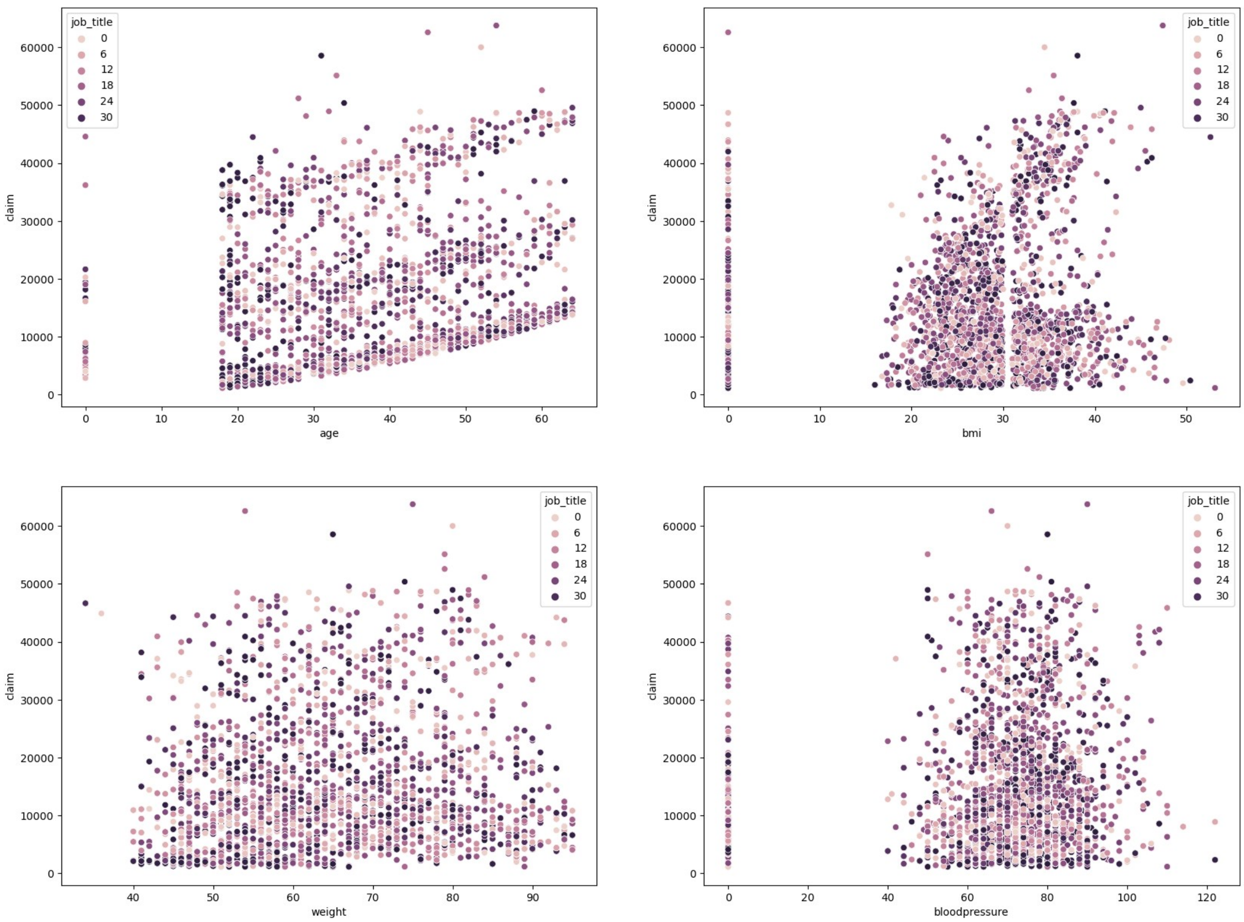
```
sex                     0.059592
weight                  0.077716
bmi                     0.098840
hereditary_diseases    -0.444337
no_of_dependents        0.067614
smoker                  0.773399
city                    0.013785
bloodpressure           0.013742
diabetes                0.135371
regular_ex             -0.060492
job_title              -0.026016
claim                   1.000000
```

```
plt.figure(figsize=(18,9))
sns.heatmap(data.corr(),annot = True, cmap ="Accent_r")
```

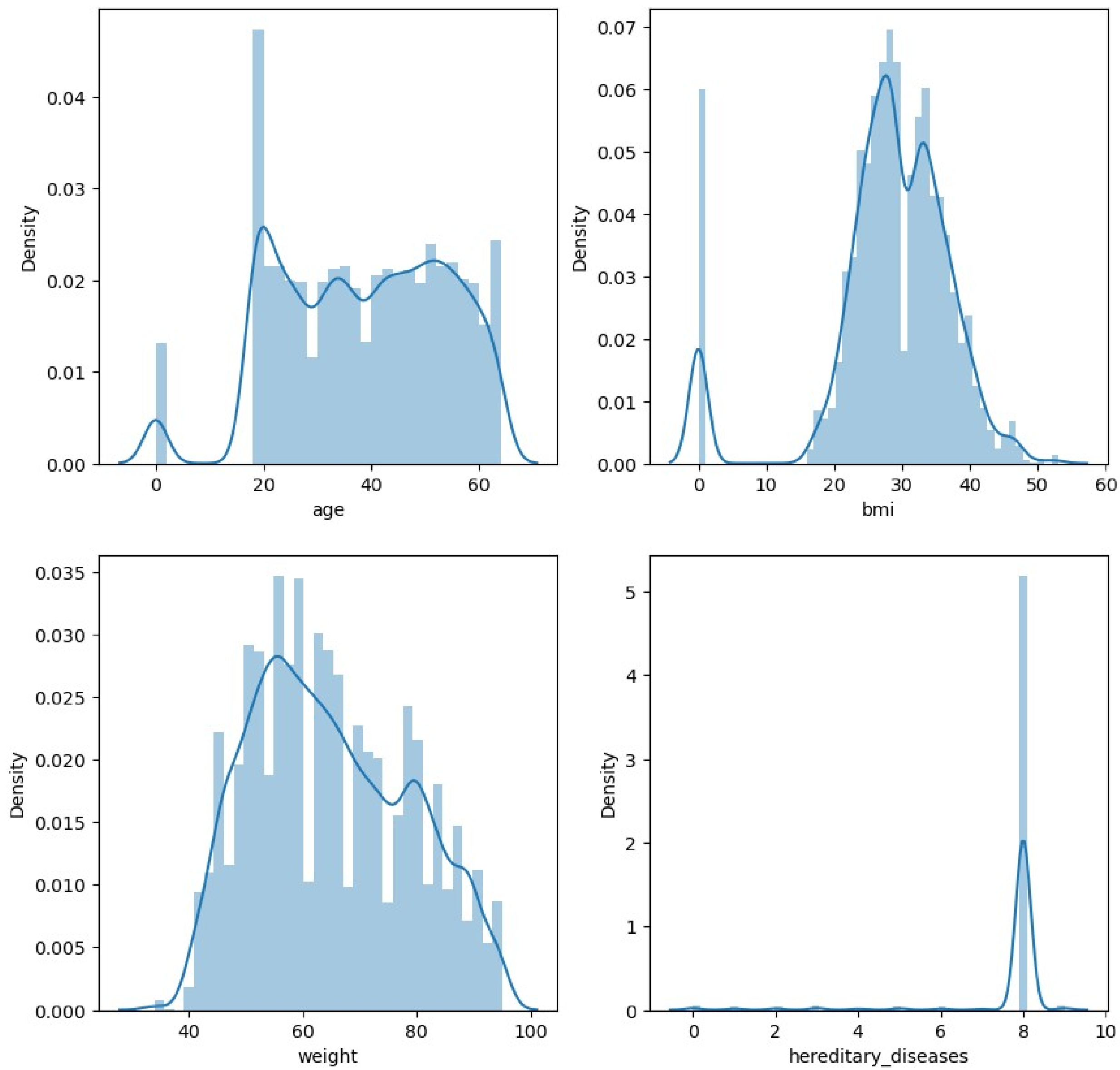
```
<AxesSubplot:>
```



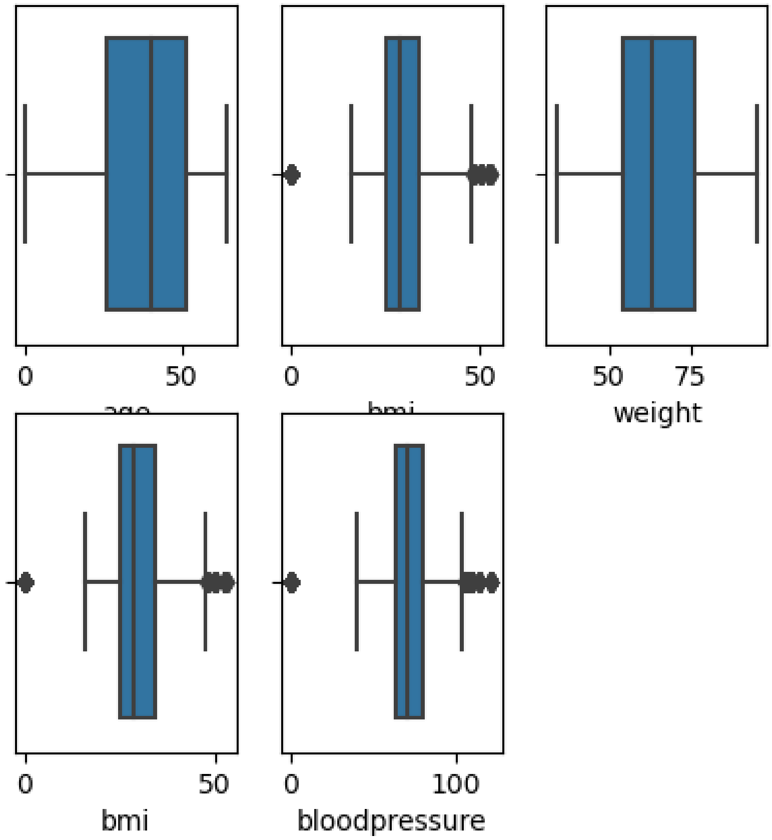

```
features = ['age','bmi','weight','bloodpressure']

plt.subplots(figsize=(20,15))

for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sns.scatterplot(data=data,x=col,y='claim',hue='job_title')
plt.show()
```



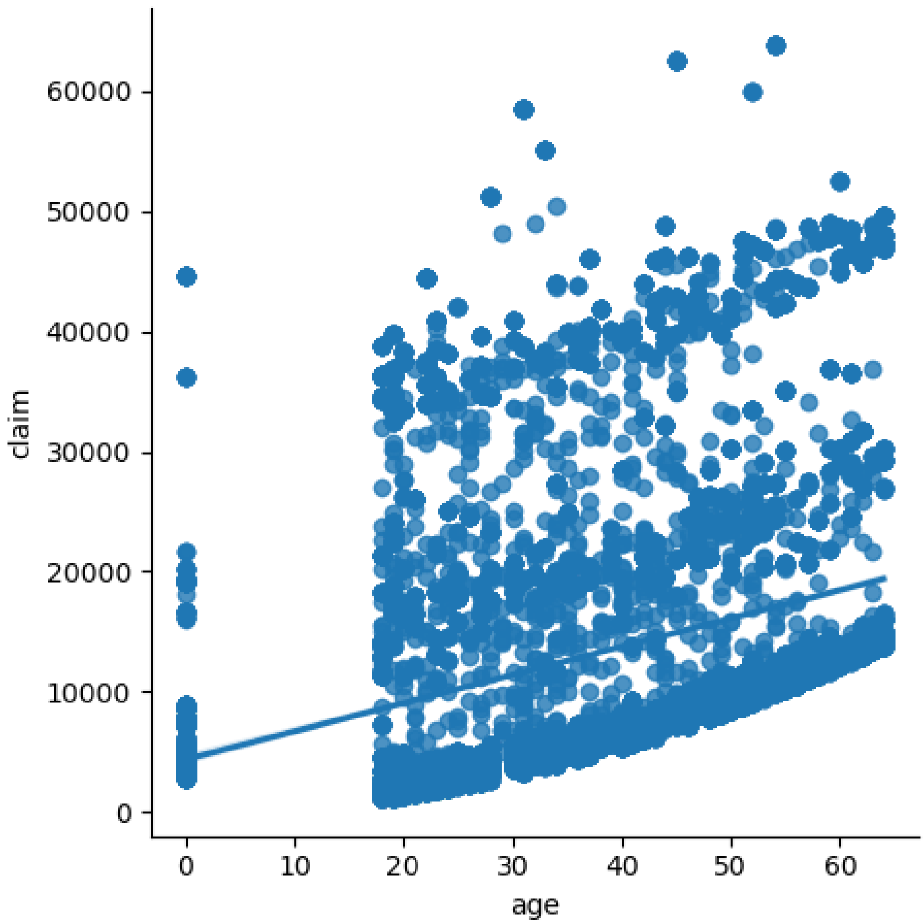

```
features = ['age','bmi','weight','hereditary_diseases']

plt.subplots(figsize=(10,10))

for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sns.distplot(data[col])
plt.show()
```




```
features = ['age','bmi','weight','bmi','bloodpressure',]

plt.subplots(figsize=(5,5))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.boxplot(data[col])
plt.show()
```

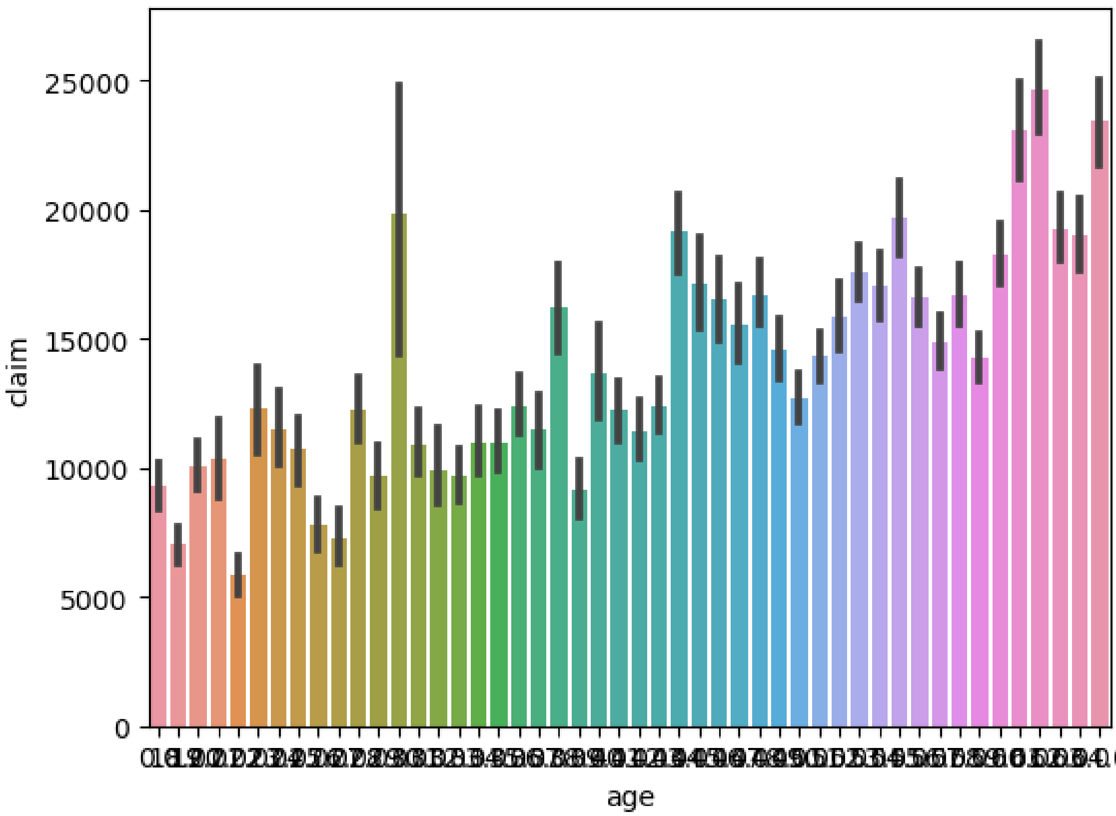



```
#sns.boxplot(x='age',y='claim',data=data)

sns.lmplot(x='age',y='claim',data=data)

<seaborn.axisgrid.FacetGrid at 0x1da5a086850>
```



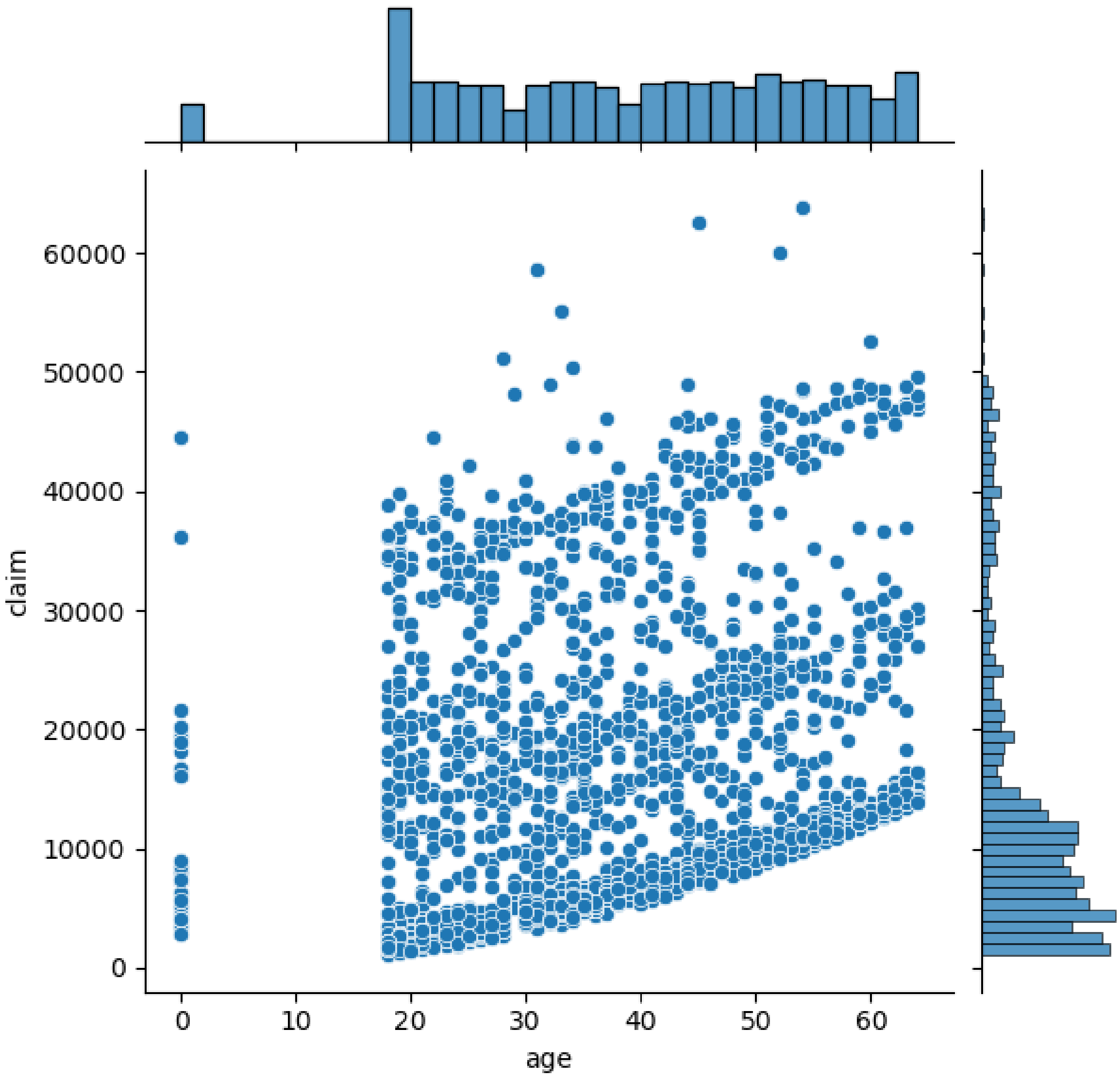

```
sns.barplot(x='age',y='claim',data=data)
```

```
<AxesSubplot:xlabel='age', ylabel='claim'>
```




```
sns.jointplot(x='age',y='claim',data=data)
```

```
<seaborn.axisgrid.JointGrid at 0x1da59fc4550>
```




```
x=data.iloc[:,0:12].values
x.shape
```

```
(15000, 12)
```

```
y=data.iloc[:,4].values
print(y.shape)
```

```
(15000,)
```

Logistic Regression

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=1)
```

```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred=reg.predict(x_test)
```

```
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print("Training Score: ",reg.score(x_train,y_train)*100)
```

```
              precision    recall  f1-score   support

           0       0.69      0.78      0.73        23
           1       0.44      0.70      0.54        23
           2       0.42      0.19      0.26        26
           3       0.40      0.35      0.38        34
           4       0.00      0.00      0.00        14
           5       0.57      0.59      0.58        27
           6       0.00      0.00      0.00        19
           7       0.00      0.00      0.00        11
           8       0.98      1.00      0.99      2798
           9       0.00      0.00      0.00        25

    accuracy                           0.95      3000
   macro avg       0.35      0.36      0.35      3000
weighted avg       0.94      0.95      0.95      3000

[[  18    4    1    0    0    0    0    0    0    0]
 [   5   16    0    1    0    1    0    0    0    0]
 [   3    4    5    8    4    1    1    0    0    0]
 [   0    9    5   12    0    7    1    0    0    0]
 [   0    3    0    3    0    3    5    0    0    0]
 [   0    0    1    6    4   16    0    0    0    0]
 [   0    0    0    0    0    0    0    7   12    0]
 [   0    0    0    0    0    0    0    0   11    0]
 [   0    0    0    0    0    0    0    0 2798    0]
 [   0    0    0    0    0    0    0    0   25    0]]
Training Score:  96.025
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

```
      Actual  Predicted
0          8          8
1          8          8
2          8          8
3          8          8
4          9          8
...      ...        ...
2995       8          8
2996       8          8
2997       8          8
2998       8          8
2999       8          8

[3000 rows x 2 columns]
```

```
print(accuracy_score(y_test,y_pred)*100)
```

```
95.5
```

```
from sklearn.model_selection import GridSearchCV
param = {
         'penalty':['l1','l2'],
         'C':[0.001, 0.01, 0.1, 1, 10, 20,100, 1000]
}
lr= LogisticRegression(penalty='l1')
cv=GridSearchCV(reg,param,cv=5,n_jobs=-1)
cv.fit(x_train,y_train)
cv.predict(x_test)
```

```
array([8, 8, 8, ..., 8, 8, 8], dtype=int64)
```

```
print("Best CV score", cv.best_score_*100)
```

```
Best CV score 96.18333333333334
```

Random Forest

```
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)



clfr=RandomForestClassifier(n_estimators=10,criterion='entropy',random
_state=0)

clfr.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10,
random_state=0)
```

```
clfr1=RandomForestClassifier(n_estimators=10,criterion='gini',random_s
tate=0)

clfr1.fit(x_train,y_train)
```

```
RandomForestClassifier(n_estimators=10, random_state=0)
```

```
ypre=clfr.predict(x_test)# entropy ypre calculation

ypre1=clfr1.predict(x_test)# gini ypre calculation

print('entropy Accuracy Score:')
accuracy_score(y_test,ypre)*100
```

```
entropy Accuracy Score:

99.83333333333333
```

```
print('gini Accuracy Score:')
accuracy_score(y_test,ypre1)*100
```

```
gini Accuracy Score:

99.8
```

```
print('entropy - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre))
print('gini - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre1))
```

```
entropy - confusion matrix
--------------------

[[  23    0    0    0    0    0    0    0    0    0]
 [   1   21    1    0    0    0    0    0    0    0]
 [   0    0   26    0    0    0    0    0    0    0]
 [   0    0    0   34    0    0    0    0    0    0]
 [   0    1    0    0   13    0    0    0    0    0]
 [   0    0    0    0    0   27    0    0    0    0]
 [   0    0    0    0    0    0   19    0    0    0]
 [   0    0    0    0    0    0    0   10    1    0]
 [   0    0    0    0    0    0    0    0 2798    0]
 [   0    0    0    0    0    0    0    0    1   24]]
gini - confusion matrix
--------------------

[[  23    0    0    0    0    0    0    0    0    0]
 [   0   23    0    0    0    0    0    0    0    0]
```

```
 [   0    1   24    1    0    0    0    0    0    0]
 [   0    0    0   34    0    0    0    0    0    0]
 [   0    0    0    2   12    0    0    0    0    0]
 [   0    0    0    0    0   27    0    0    0    0]
 [   0    0    0    0    0    0   19    0    0    0]
 [   0    0    0    0    0    0    1   10    0    0]
 [   0    0    0    0    0    0    0    0 2798    0]
 [   0    0    0    0    0    0    0    0    1   24]]
```

```python
print('entropy result\n--------------')
print(classification_report(y_test,ypre))
print('gini index result\n---------------------')
print(classification_report(y_test,ypre1))
```

```
entropy result
--------------
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        23
           1       0.95      0.91      0.93        23
           2       0.96      1.00      0.98        26
           3       1.00      1.00      1.00        34
           4       1.00      0.93      0.96        14
           5       1.00      1.00      1.00        27
           6       1.00      1.00      1.00        19
           7       1.00      0.91      0.95        11
           8       1.00      1.00      1.00      2798
           9       1.00      0.96      0.98        25

    accuracy                           1.00      3000
   macro avg       0.99      0.97      0.98      3000
weighted avg       1.00      1.00      1.00      3000

gini index result
---------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       0.96      1.00      0.98        23
           2       1.00      0.92      0.96        26
           3       0.92      1.00      0.96        34
           4       1.00      0.86      0.92        14
           5       1.00      1.00      1.00        27
           6       0.95      1.00      0.97        19
           7       1.00      0.91      0.95        11
           8       1.00      1.00      1.00      2798
           9       1.00      0.96      0.98        25

    accuracy                           1.00      3000
   macro avg       0.98      0.96      0.97      3000
```

```
weighted avg       1.00      1.00      1.00      3000
```

Decision Tree

```
dtree = DecisionTreeClassifier(max_depth=6, random_state=1)

dtree.fit(x_train,y_train)

DecisionTreeClassifier(max_depth=6, random_state=1)

y_pred=dtree.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print("Training Score: ",dtree.score(x_train,y_train)*100)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       1.00      1.00      1.00        23
           2       1.00      1.00      1.00        26
           3       1.00      1.00      1.00        34
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        27
           6       1.00      1.00      1.00        19
           7       1.00      1.00      1.00        11
           8       1.00      1.00      1.00      2798
           9       1.00      1.00      1.00        25

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

[[  23    0    0    0    0    0    0    0    0    0]
 [   0   23    0    0    0    0    0    0    0    0]
 [   0    0   26    0    0    0    0    0    0    0]
 [   0    0    0   34    0    0    0    0    0    0]
 [   0    0    0    0   14    0    0    0    0    0]
 [   0    0    0    0    0   27    0    0    0    0]
 [   0    0    0    0    0    0   19    0    0    0]
 [   0    0    0    0    0    0    0   11    0    0]
 [   0    0    0    0    0    0    0    0 2798    0]
 [   0    0    0    0    0    0    0    0    0   25]]
Training Score:  100.0
```

```
print(accuracy_score(y_test,y_pred)*100)
```

```
100.0
```

Naive Bayes Theorem

```
Gmodel=GaussianNB() # invocation

Gmodel.fit(x_train,y_train)

train_Gpred=Gmodel.predict(x_train)

test_Gpred=Gmodel.predict(x_test)

train_acc_gau=np.mean(train_Gpred==y_train)

test_acc_gau=np.mean(test_Gpred==y_test)
print('gaussian naive bayes - training 
accuracy:',train_acc_gau*100,'%')
print('gaussian nb - testing time accuracy:',test_acc_gau*100,'%')
```

```
gaussian naive bayes - training accuracy: 100.0 %
gaussian nb - testing time accuracy: 100.0 %
```