感悟 2025年7月13日

你可以通过阅读某个领域，比如phishing detection base on ML 的survey，来很快地了解一个领域的所有相关使用的模型，甚至是模型的组合配合。

做ML，有从数据feature本身入手的。比如将原内容转化为lexical feature，例如将URL给抽象成high level的features，再用这个抽象的feature去进行训练数据，使得模型能够对各种varient的URL有检测的能力。

也有从模型的组合搭配上，进行操作。比如同时用多个模型预测同一组features，将它们各自的分类结果，综合起来，给出最终的答案。

感悟 2025年5月5日

我现在已经大致了解mining data的全流程，以及对应的loss函数在训练模型中的实际作用。

loss函数的选择，比如**mae, mse, rmse**等，取决于数据本身的情况，如果数据是high dynamic range的（30到1080，甚至更高），那就选择mse，或者rmse，因为当预测的值与实际相差时，它还会做一个平方，这就使得loss非常地大，而model的训练则是要尽可能地减少loss，所以最终该模型的输出，会很适合预测出高离散时的value。当然，还有优化函数的选择，比如adam。

所以**整体大模型训练的流程**就是，你有一个想研究的目标，你获取相关目标的数据，你根据数据的特性，情况，来选择合适的模型。同时也根据数据本身数值的情况，进行**预处理**，如果是普通的正态分布，还可以用standard scaler或者不用。如果是hign dynamic range value，那就选MinMaxScaler来转换。选择你想要的feature，来预测target feature。再用训练好的模型，去预测新数据，得到可信度较高的结果。

而模型的不同，适用于不同的功能，主要就两类，要么**分类**（比如聚类，图像识别，语音识别，filter等），要么**预测**（预测未来的value，协同过滤等，LSTM，RNN等）。

总步骤

第一步： 找到需要研究的数据，以及想要了解的问题。比如你想预测的target feature是哪一个?

第二步： 数据清洗 (缺失数据处理，重命名columns, cut数据分类，分类数据转数值数据，strip各项feature中的value值。**)**

第三步：数据visualizing(获得categorical, numerical features, countplot, boxplot, plot.pie, heatmap等**)**

第四步：数据训练的预处理

使用**label encoder**，将categorical 数据转化成配对的numerical数据。

数据的标准化**normalizing**，也就是scaler，将除了target features之外的所有数据转化成[0,1]。所有scaler的数据是作为模型的input。

处理**imbalanced**数据，也就是每个feature中values的样本平衡。比如income feature，>50k的数量远远大于 <=50k的样本数量，就需要去平衡它们的数量。

注意，不用对target feature(你要预测的feature)做归一化。只需要输入给模型的features 进行normalizing即可。

第五步: 建立模型，开始训练

不同的模型有不同的作用，比如常见的**分类模型**（Logistic regression, naive bayes, SVM等），如果要**预测数值**的话呢，则可以用linearRegression.

第六步：评估预测结果

使用cross_val_score，交叉验证，得出模型预测不同数据的平均分。

📌 原因二：提升评估稳定性

交叉验证会：

- 自动把数据分成 `cv` 份（例如 `cv=5`）
- 每次选一份做验证，其余做训练
- 算出多个模型表现，再**取平均得分**

这样可以有效避免模型在**某一份数据上表现特别好或差**，评估更全面。

具体操作

为数据做**清洗**，比如将NA值替换成该属性的平均值，如age属性

1. label encoding （看情况做。）

将**分类数值，替换成数字类型**...

change categorical feature into numerical data.

[174]:
```
from sklearn.preprocessing import LabelEncoder
```

[175]:
```
#change categorical features into numerical.
def label_encoder(a):
    le = LabelEncoder()
    #Fit the encoder
    df[a]=le.fit_transform(df[a])
```

[180]:
```
df.head()
```

[180]:

weight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	BMI	FCVC_types	NCP_types
yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	Normal_Weight	24.386526	mid	three
yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight	24.238227	high	three
yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	Normal_Weight	23.765432	mid	three
no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Overweight_Level_I	26.851852	high	three
no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Overweight_Level_II	28.342381	mid	one

```
[178]: categorical

[178]: ['Gender',
        'Family_history_with_overweight',
        'FAVC',
        'CAEC',
        'SMOKE',
        'SCC',
        'CALC',
        'MTRANS',
        'NObeyesdad']

[182]: for c in categorical:
           label_encoder(c)

[184]: df.head()
```

[184]:

	Gender	Age	Height	Weight	Family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	
0	0	21.0	1.62	64.0	1	0	2.0	3.0	2	0	2.0	0	0.0	1.0	3	3	1	24
1	0	21.0	1.52	56.0	1	0	3.0	3.0	2	1	3.0	1	3.0	0.0	2	3	1	24
2	1	23.0	1.80	77.0	1	0	2.0	3.0	2	0	2.0	0	2.0	1.0	1	3	1	23
3	1	27.0	1.80	87.0	0	0	3.0	3.0	2	0	2.0	0	2.0	0.0	1	4	5	26
4	1	22.0	1.78	89.8	0	0	2.0	1.0	2	0	2.0	0	0.0	0.0	2	3	6	28

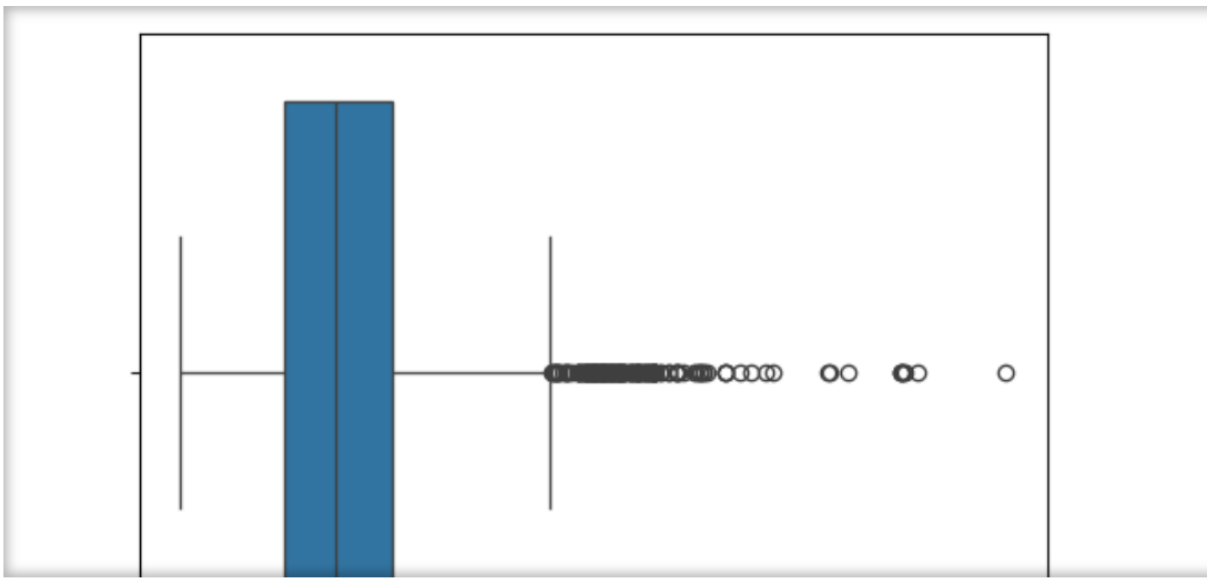
2. 更改column 名

取得catergorical 和Numerical 的数据features，并做对应的视觉化展示

```
[41]: #find catergorical features.
      categorical=[var for var in df.columns if df[var].dtype=='O']
      categorical

[41]: ['Gender',
       'Family_history_with_overweight',
       'FAVC',
       'CAEC',
       'SMOKE',
       'SCC',
       'CALC',
       'MTRANS',
       'NObeyesdad']

[45]: #find numerical features.
      numerical=[]
      for i in df.columns:
          if i not in categorical:
              numerical.append(i)
      numerical

[45]: ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE', 'BMI']
```

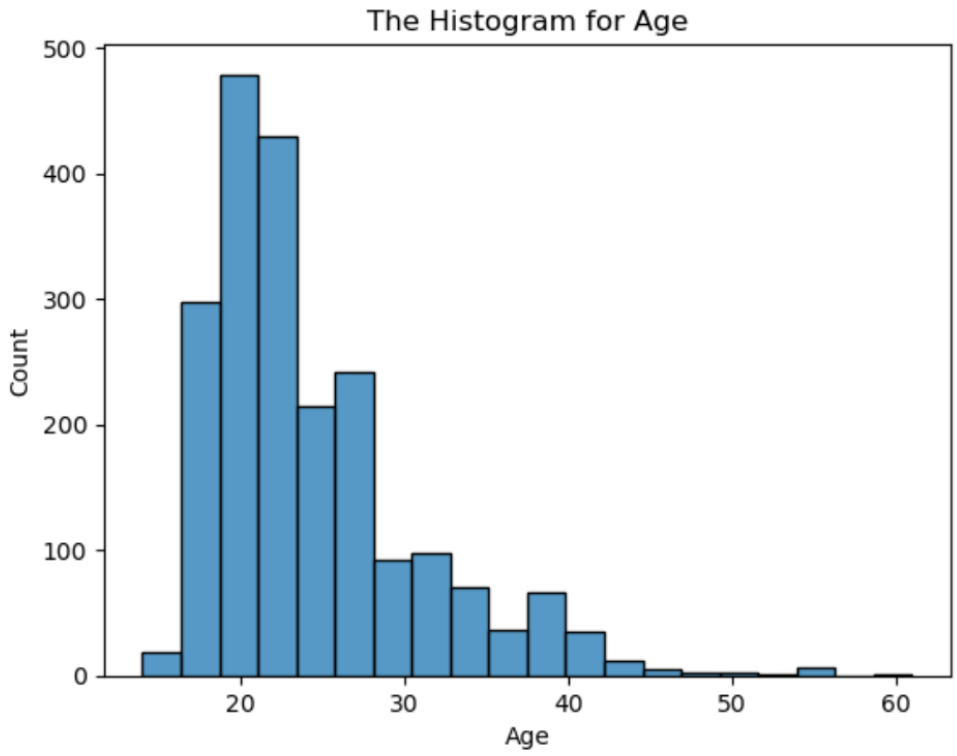
对于numerical数据，可以用boxplot和histogram（柱状图）

[139]:
```
for n in numerical:
    sns.boxplot(x=n,data=df)
    plt.show()
```



[141]:
```
for i in numerical:
    sns.histplot(x=df[i], palette='Set1',bins=20)#分成20个柱子。
    plt.title("The Histogram for {}".format(i)) #柱状图
    plt.show()
```

```
C:\Users\10929\AppData\Local\Temp\ipykernel_19280\3151178761.py:2: UserWarning: Ignor
d.
  sns.histplot(x=df[i], palette='Set1',bins=20)#分成20个柱子。
```



categorical 类的数据，则用countplot 来每一个feature配对。

```
[171]: #produce 8 row and 1 column of the whole graph because we enter 8 features of categorical values.
fig, axs = plt.subplots(8, 1, figsize=(15,20))

axs = axs.flatten()
fig.suptitle('Relation between the categorical features and income')
#8个features
categorical2 = ['Gender',
 'Family_history_with_overweight',
 'FAVC',
 'CAEC',
 'SMOKE',
 'SCC',
 'CALC',
 'MTRANS']
for ax, i in enumerate(categorical2):
    sns.countplot(x='NObeyesdad', alpha=0.7, hue=i, data=df, ax=axs[ax],palette='Set1')
```

3. 数据的数值分类，从quantitative到categorical的转换，或相反

4. 某列数据中的字符需要strip掉多余空格或符号

5. 将categorical和quantitative类型的列给区分开，为后续的可视化做准备

6. normalization

不要对target feature（你要预测的feature）进行标准化。

💡 为什么不对目标特征进行标准化?

1. 目标值不参与模型的输入

标准化的主要目的是帮助模型更好地学习 —— 它是为输入特征服务的。我们希望所有输入特征落在相似的数值范围（比如均值为0，方差为1），这样模型在训练时更容易收敛，梯度不会因为特征尺度差异而不平衡。

> 但目标值（比如 income）是我们要预测的结果，本身不会参与模型的输入特征中。

2. 模型要学的是原始目标值的模式

如果你对目标变量进行了标准化，那模型学到的其实是"标准化后的目标"，这意味着你**必须在预测后还原回原始值（反标准化）**才能有意义。虽然可以这么做，但通常没必要，除非模型对数值范围特别敏感（比如某些回归任务中有极端的数值波动）。

3. 保持评估指标的可读性

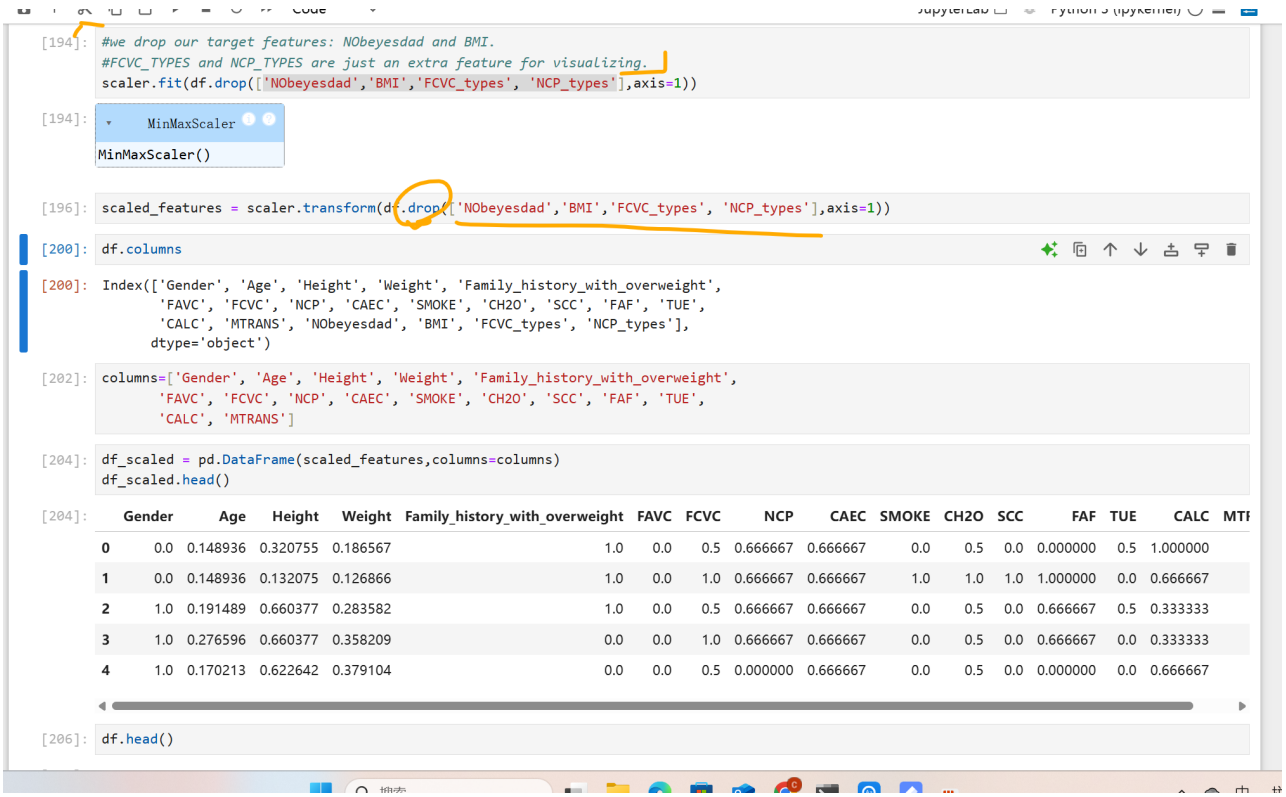
你想要预测的是原始的收入（income），那就保留它的原始形式，这样你用例如 MSE、RMSE、MAE 等指标评估预测结果时，结果也才是"以元为单位"的，而不是标准化后的单位，否则结果就难以解释了。

note:-不需要把target features数据给进行归一化，下图中的训练的**target数据就是income**，除了income数据和其他全0数据外，其他都进行标准化。

有许多种normalizing的方法，minMaxScaler是其中一种。

将**数据进行归一，统一尺度，提升模型训练的质量。**

所以，在进行模型训练的时候，将scale后的数据作为输入input，而将原数据中的target feature作为要预测的feature.



```
[194]: #we drop our target features: NObeyesdad and BMI.
       #FCVC_TYPES and NCP_TYPES are just an extra feature for visualizing.
       scaler.fit(df.drop(['NObeyesdad','BMI','FCVC_types', 'NCP_types'],axis=1))

[194]: MinMaxScaler
       MinMaxScaler()

[196]: scaled_features = scaler.transform(df.drop(['NObeyesdad','BMI','FCVC_types', 'NCP_types'],axis=1))

[200]: df.columns

[200]: Index(['Gender', 'Age', 'Height', 'Weight', 'Family_history_with_overweight',
              'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
              'CALC', 'MTRANS', 'NObeyesdad', 'BMI', 'FCVC_types', 'NCP_types'],
             dtype='object')

[202]: columns=['Gender', 'Age', 'Height', 'Weight', 'Family_history_with_overweight',
                'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
                'CALC', 'MTRANS']

[204]: df_scaled = pd.DataFrame(scaled_features,columns=columns)
       df_scaled.head()
```

[204]:

	Gender	Age	Height	Weight	Family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTI
0	0.0	0.148936	0.320755	0.186567	1.0	0.0	0.5	0.666667	0.666667	0.0	0.5	0.0	0.000000	0.5	1.000000	
1	0.0	0.148936	0.132075	0.126866	1.0	0.0	1.0	0.666667	0.666667	1.0	1.0	1.0	1.000000	0.0	0.666667	
2	1.0	0.191489	0.660377	0.283582	1.0	0.0	0.5	0.666667	0.666667	0.0	0.5	0.0	0.666667	0.5	0.333333	
3	1.0	0.276596	0.660377	0.358209	0.0	0.0	1.0	0.666667	0.666667	0.0	0.5	0.0	0.666667	0.0	0.333333	
4	1.0	0.170213	0.622642	0.379104	0.0	0.0	0.5	0.000000	0.666667	0.0	0.5	0.0	0.000000	0.0	0.666667	

```
[206]: df.head()
```

[206]:

t	Family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	BMI	FCVC_types	NCP_types
)	1	0	2.0	3.0	2	0	2.0	0	0.0	1.0	3	3	1	24.386526	mid	three
)	1	0	3.0	3.0	2	1	3.0	1	3.0	0.0	2	3	1	24.238227	high	three
)	1	0	2.0	3.0	2	0	2.0	0	2.0	1.0	1	3	1	23.765432	mid	three
)	0	0	3.0	3.0	2	0	2.0	0	2.0	0.0	1	4	5	26.851852	high	three
}	0	0	2.0	1.0	2	0	2.0	0	0.0	0.0	2	3	6	28.342381	mid	one

如何对数据进行normalization?

数据归一化（Normalization）是数据预处理的一种常用方法，它将不同尺度的数据映射到同一尺度，通常是[0, 1]或[-1, 1]。这有助于提高机器学习算法的训练效果，尤其是对于基于梯度下降的优化算法。以下是几种常见的归一化方法：

1. 最小-最大归一化（Min-Max Normalization）

将数据线性映射到[0, 1]区间。公式如下：

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

其中 x 是原始数据，$x_{\min}$ 和 $x_{\max}$ 分别是数据的最小值和最大值。

2. Z-score标准化（Z-score Normalization）

将数据标准化为均值为0，标准差为1的分布。公式如下：

$$x' = \frac{x - \mu}{\sigma}$$

其中 x 是原始数据，μ 是数据的均值，σ 是数据的标准差。

3. 均值归一化（Mean Normalization）

将数据线性映射到[-1, 1]区间。公式如下：

$$x' = \frac{x - \mu}{x_{\max} - x_{\min}}$$

其中 x 是原始数据，μ 是数据的均值，$x_{\min}$ 和 $x_{\max}$ 分别是数据的最小值和最大值。

额外一步 --- 处理imbalanced 数据

处理target feature中各value类型的数据。

Using over sampling method to handle imbalanced dataset

[70]:
```
df.head()
```

[70]:

	age	workclass	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	in
0	39	6	9	13	4	0	1	4	1	2174	0	40	38	
1	50	5	9	13	2	3	0	4	1	0	0	13	38	
2	38	3	11	9	0	5	1	4	1	0	0	40	38	
3	53	3	1	7	2	5	0	2	1	0	0	40	38	
4	28	3	9	13	2	9	5	2	0	0	0	40	4	

[220]:
```
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import NearMiss
```

[222]:
```
X = df_scaled
y= df.income
```

[224]:
```
# Implementing Oversampling for Handling Imbalanced
smk = SMOTETomek(random_state=42)
X_res,y_res=smk.fit_resample(X,y)
```

[58]:
```
X_res.shape,y_res.shape
```

[58]:
```
((48178, 13), (48178,))
```

resample后，income中的0与1数据集数量一样。

[232]:
```
from collections import Counter
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

```
Original dataset shape Counter({0: 24717, 1: 7682})
Resampled dataset shape Counter({0: 24089, 1: 24089})
```

[236]:
```
y_res.shape,y.shape
```

[236]:
```
((48178,), (32399,))
```

原，income

resample

注意事项

- 归一化应在训练数据和测试数据上分别进行，以防止数据泄露。
- **对于某些特定的算法，如决策树和随机森林，归一化可能不是必要的，因为这些算法对数据的尺度不敏感。**

- 在对数据进行归一化后，应保存归一化的参数（如均值和标准差，最小值和最大值），以便在处理新数据时使用相同的参数进行归一化。

- **第三步- 想要研究的关系**

将数据中，想要研究的某类属性与整体数据做关联的数据，做分割。

比如**Y是死亡情况，X是除了死亡情况外的所有属性数据**

```
In [17]: #这一步的目的是研究，乘客的年龄，level，性别...与死亡率的关系
         Y=Data2.Survived #直接获取某列数据
         X=Data2
         X.drop(['Survived'],axis=1,inplace=True) #axis=0 表示要去除的是row， axis=
         #train_test_split是skilearn的方法。
```

- **第四步- 训练数据与测试数据划分**

将数据划分成**训练数据**和**测试数据**

一般可以使用**scikit** 包，使用'train_test_split'和各类模型

```
1 X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.2, random_state=3)
```

```
In [27]: #训练模型时，至少是同行数row
         #一组数据与某一维度的性质的数据，做关联。
         print(X_train.shape,Y_train.shape)
         print(X_test.shape,Y_test.shape)
```

```
(712, 5) (712,)
(179, 5) (179,)
```

- **第五步，选择模型，开始训练**

import相关的模型

```
In [1]: from sklearn.model_selection import train_test_split
        from sklearn.metrics import precision_recall_fscore_support, accuracy_score,classification_report
        import pandas as pd
        import seaborn as sns
        import numpy as np

        from sklearn.naive_bayes import BernoulliNB
        from sklearn import tree
        from sklearn.neighbors import KNeighborsClassifier
        %matplotlib inline
        import matplotlib.pyplot as plt

        Data=pd.read_csv('titanic.csv',index_col=0)
        # todo: display the head of the data using pandas
```

naive 贝叶斯分类器

In [21]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html
#分类器，用的是pure贝叶斯
#alpha 表示平滑参数，0表示不平滑。
#fit_prior 是否学习类先验概率。如果为 false，则将使用统一先验。
classifier=BernoulliNB(fit_prior=True, alpha=1.0)
classifier.fit(X_train, Y_train)#训练数据
predicted_y=classifier.predict(X_test)
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
#判断实际与预测结果的情况
#zero_divisionSets the value to return when there is a zero division. If set to “warn”, this acts as
print(classification_report(Y_test, predicted_y,zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.84      0.81      0.82       109
           1       0.72      0.76      0.74        70

    accuracy                           0.79       179
   macro avg       0.78      0.78      0.78       179
weighted avg       0.79      0.79      0.79       179
```

这个fit，就是为了训练模型，以**Y死亡情况**和**X乘客的各项属性**，做关联，训练好后。

我们就可以用模型去预测其他乘客的死亡情况。

决策树模型

In [30]:
```
from sklearn.tree import plot_tree
#模型选择—决策树
tree_classifier = tree.DecisionTreeClassifier(min_samples_leaf=1,
    min_samples_split=2, max_depth=None,criterion='entropy',random_state=0)
#开始训练
tree_classifier.fit(X_train,Y_train)
tree_predict=tree_classifier.predict(X_test)
# todo predict survival for the test part and print results
print(classification_report(Y_test,tree_predict,zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.83      0.79      0.81       109
           1       0.69      0.74      0.72        70

    accuracy                           0.77       179
   macro avg       0.76      0.77      0.76       179
weighted avg       0.77      0.77      0.77       179
```

KNN分类器

In [31]:
```
#knn 分类器
# p is the parameter for Minkowski distance
# weights can be also 'distance'
# todo implement k-nn with 5 neighbours, predict again and print metrics
knn_classifier=KNeighborsClassifier(n_neighbors=5,p=2)
knn_classifier.fit(X_train,Y_train)
knn_predict=knn_classifier.predict(X_test)
print(classification_report(Y_test,knn_predict,zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.76      0.83      0.79       109
           1       0.69      0.59      0.64        70

    accuracy                           0.74       179
   macro avg       0.73      0.71      0.72       179
weighted avg       0.73      0.74      0.73       179
```

图片显示代码

如何使用图片展示工具

图片显示packages.

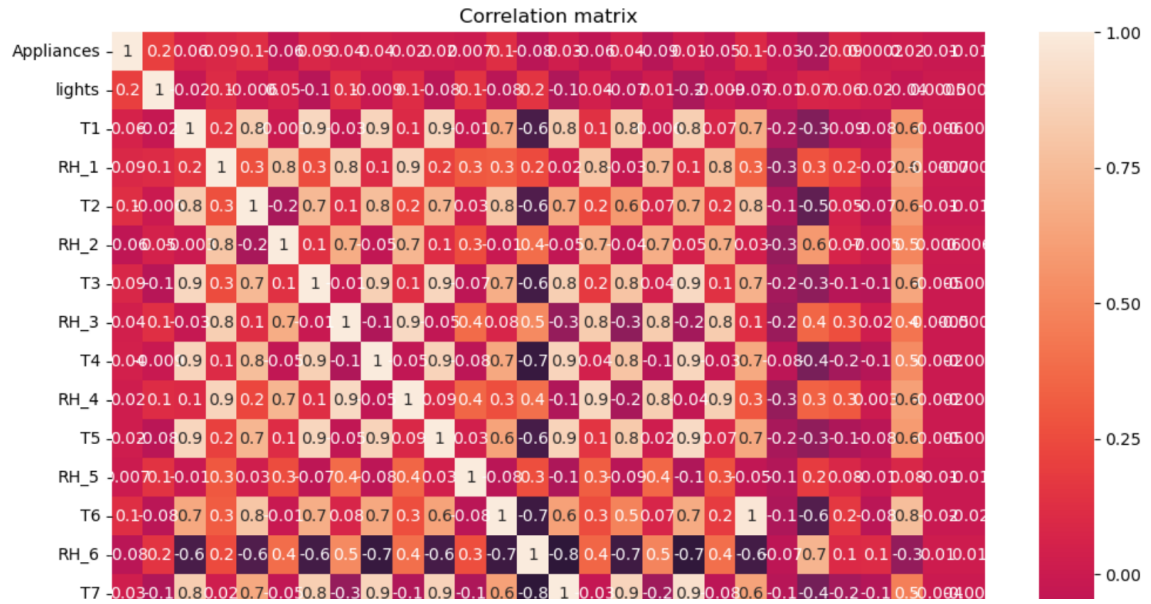
```
import matplotlib.pyplot as plt
import seaborn as sns
```

heatmap

图片的大小设置要写在制图command前面。

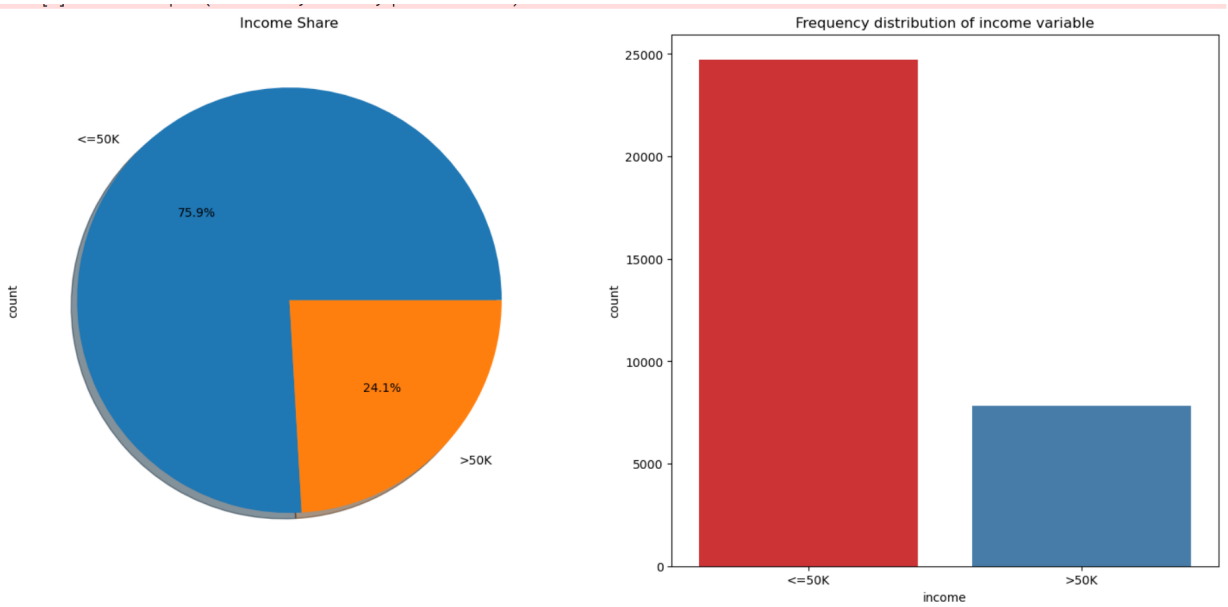
```
corr = df.corr()
#matrix = np.triu(corr)
plt.figure(figsize=(12,12))
sns.heatmap(corr, vmax=1.0, vmin=-1.0, fmt='.1g', annot=True)
plt.title('Correlation matrix')
plt.show()
```

```
plt.show()
```



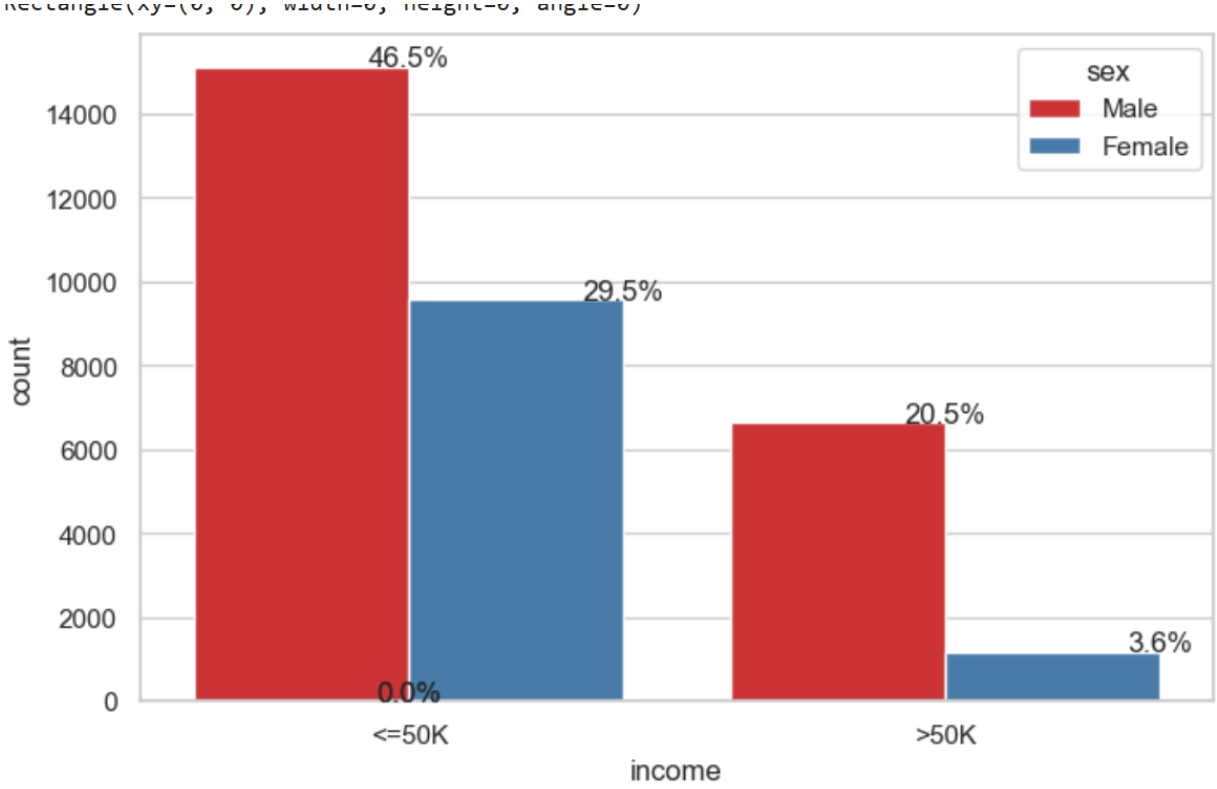
pie chart and bar chart

```
f,ax=plt.subplots(1,2,figsize=(18,8))

ax[0] = df['income'].value_counts().plot.pie(explode=
[0,0],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Income Share')


#f, ax = plt.subplots(figsize=(6, 8))
ax[1] = sns.countplot(x="income", data=df, palette="Set1")
ax[1].set_title("Frequency distribution of income variable")

plt.show()
```



柱状图使用

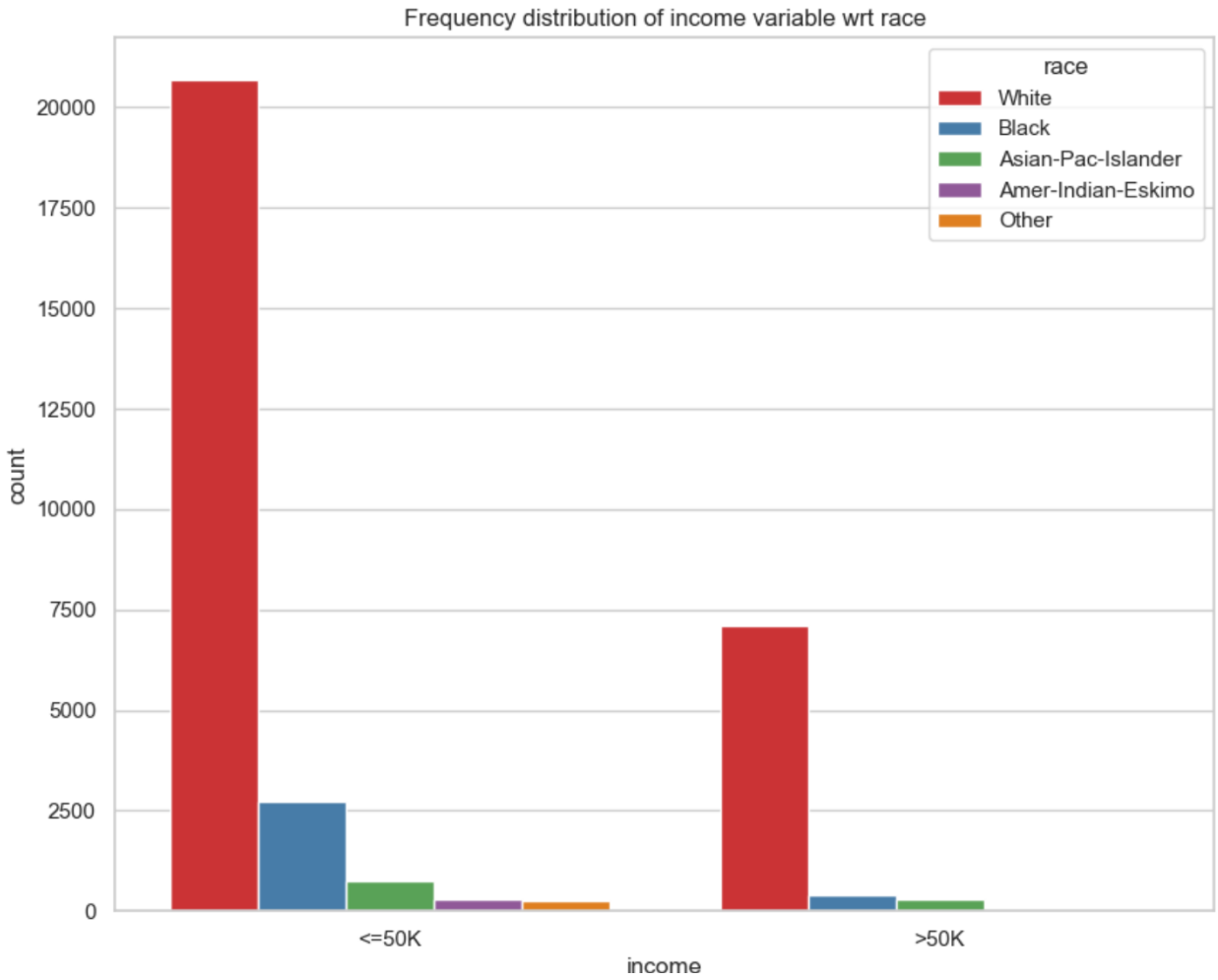
```
sns.set(style="whitegrid")#图片限制风格
plt.figure(figsize=(8,5))#设置图片大小
total = float(len(df))
ax = sns.countplot(x="income", hue="sex", data=df,palette='Set1')
# print(ax.patches)
#plt.title('No. of Smokers', fontsize=20)
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    #设置百分比符号放在图中的那个位置。
    x = p.get_x() +p.get_width()
    y = p.get_height()
    ax.annotate(percentage, (x, y),ha='center')
    print(p)#每一个p，就是每一个bar的高度和宽度信息。
# # plt.show()
```



多重柱状图

hue是分类的数据。

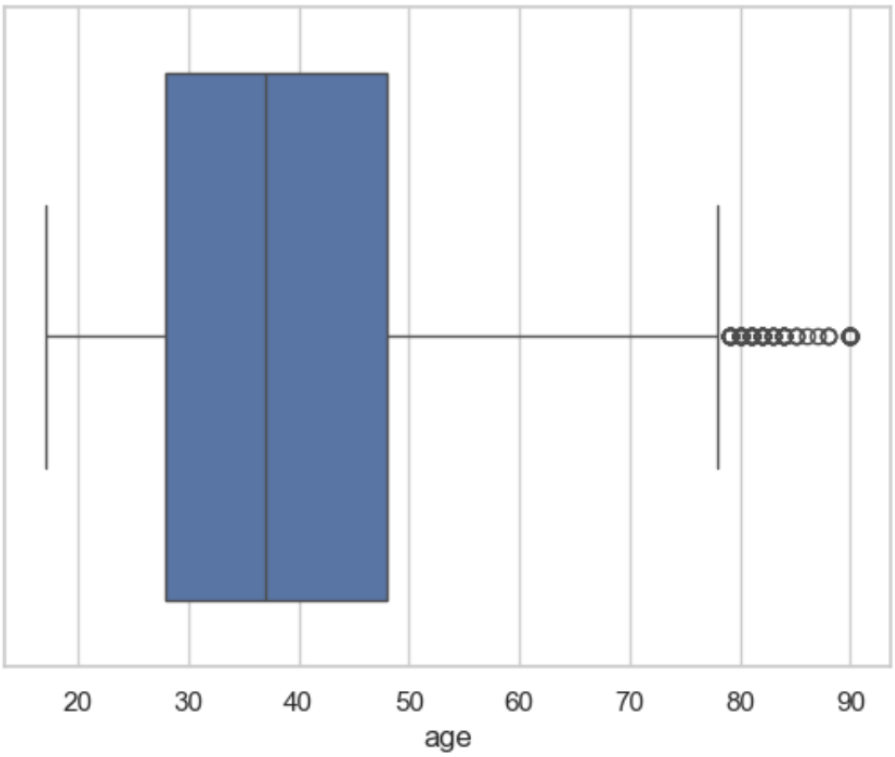
```
f, ax = plt.subplots(figsize=(10, 8))
ax = sns.countplot(x="income", hue="race", data=df, palette="Set1")
ax.set_title("Frequency distribution of income variable wrt race")
plt.show()
```


Frequency distribution of income variable wrt race
race
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
count
20000
17500
15000
12500
10000
7500
5000
2500
0
<=50K
>50K
income

boxplot

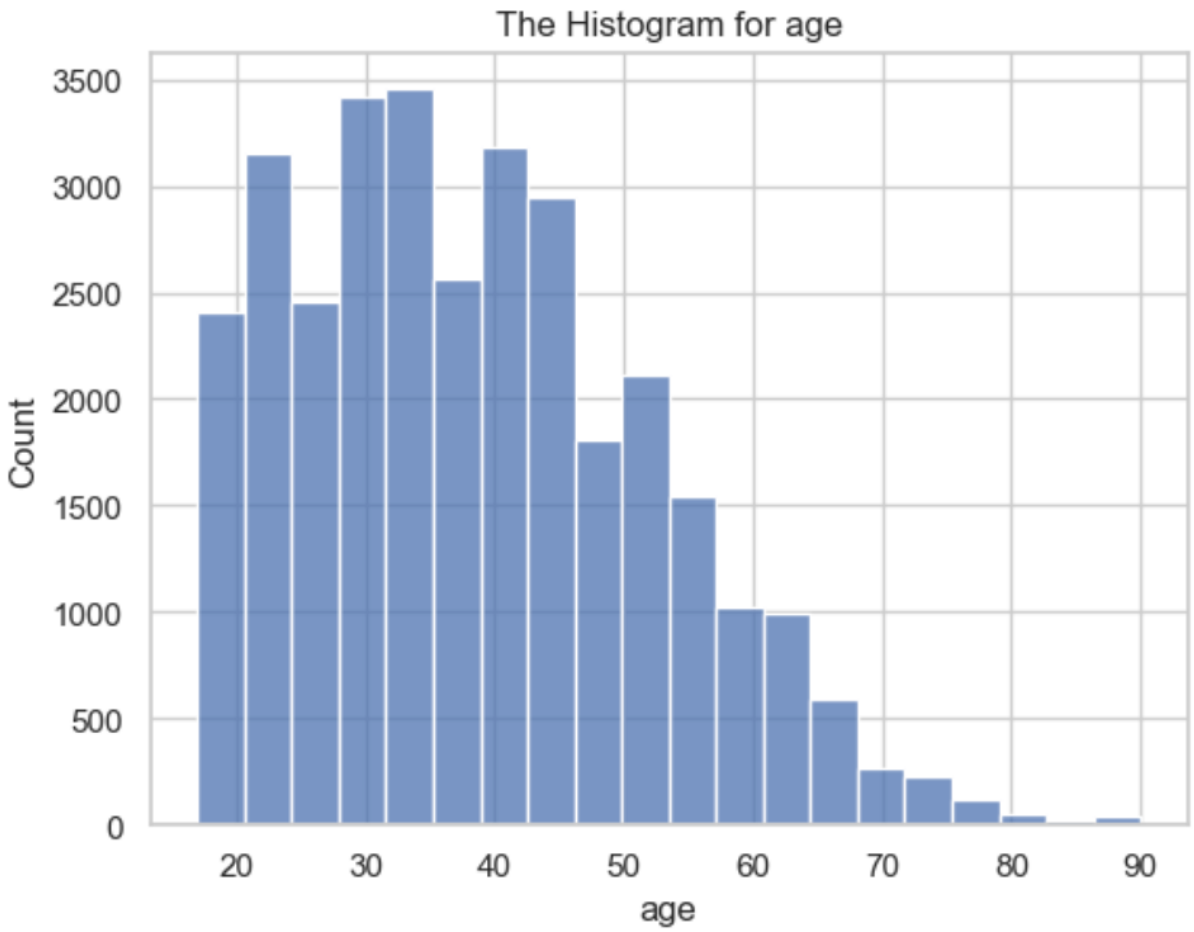
[112]:
```
sns.boxplot(x='age',data=df)
```

[112]: <Axes: xlabel='age'>



histplot

```
for i in numerical:
    sns.histplot(x=df[i], palette='Set1',bins=20)#分成20个柱子。
    plt.title("The Histogram for {}".format(i)) #柱状图
    plt.show()
```


The Histogram for age
Count
3500
3000
2500
2000
1500
1000
500
0
20
30
40
50
60
70
80
90
age