In [1]:
```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\rajesh\anaconda3\lib\site-packages (2.4.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\rajesh\anaconda3\lib\site-packages (from category_encoder
s) (1.7.1)
Requirement already satisfied: patsy>=0.5.1 in c:\users\rajesh\anaconda3\lib\site-packages (from category_encoder
s) (0.5.2)
Requirement already satisfied: pandas>=0.21.1 in c:\users\rajesh\anaconda3\lib\site-packages (from category_encod
ers) (1.3.4)
Requirement already satisfied: numpy>=1.14.0 in c:\users\rajesh\anaconda3\lib\site-packages (from category_encode
rs) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\rajesh\anaconda3\lib\site-packages (from category
_encoders) (0.24.2)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\rajesh\anaconda3\lib\site-packages (from category_e
ncoders) (0.12.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\rajesh\anaconda3\lib\site-packages (from pandas
>=0.21.1->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\rajesh\anaconda3\lib\site-packages (from pandas>=0.21.1->
category_encoders) (2021.3)
Requirement already satisfied: six in c:\users\rajesh\anaconda3\lib\site-packages (from patsy>=0.5.1->category_en
coders) (1.15.0)
Requirement already satisfied: joblib>=0.11 in c:\users\rajesh\anaconda3\lib\site-packages (from scikit-learn>=0.
20.0->category_encoders) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\rajesh\anaconda3\lib\site-packages (from scikit-l
earn>=0.20.0->category_encoders) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")
```

In [3]:
```
import category_encoders as ce
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [4]:
```
data=pd.read_csv("Company_Data.csv")
data.head()
```

Out[4]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No

In [5]:
```
data.shape
```

Out[5]:
```
(400, 11)
```

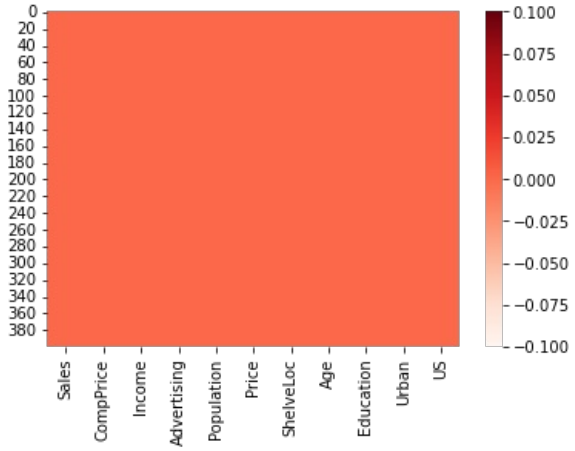
In [6]:
```
data.describe().round(2).style.background_gradient(cmap = 'Oranges')
```

Out[6]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Education
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	7.500000	124.980000	68.660000	6.640000	264.840000	115.800000	53.320000	13.900000
std	2.820000	15.330000	27.990000	6.650000	147.380000	23.680000	16.200000	2.620000
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	10.000000
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	12.000000
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	14.000000
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	16.000000
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	18.000000

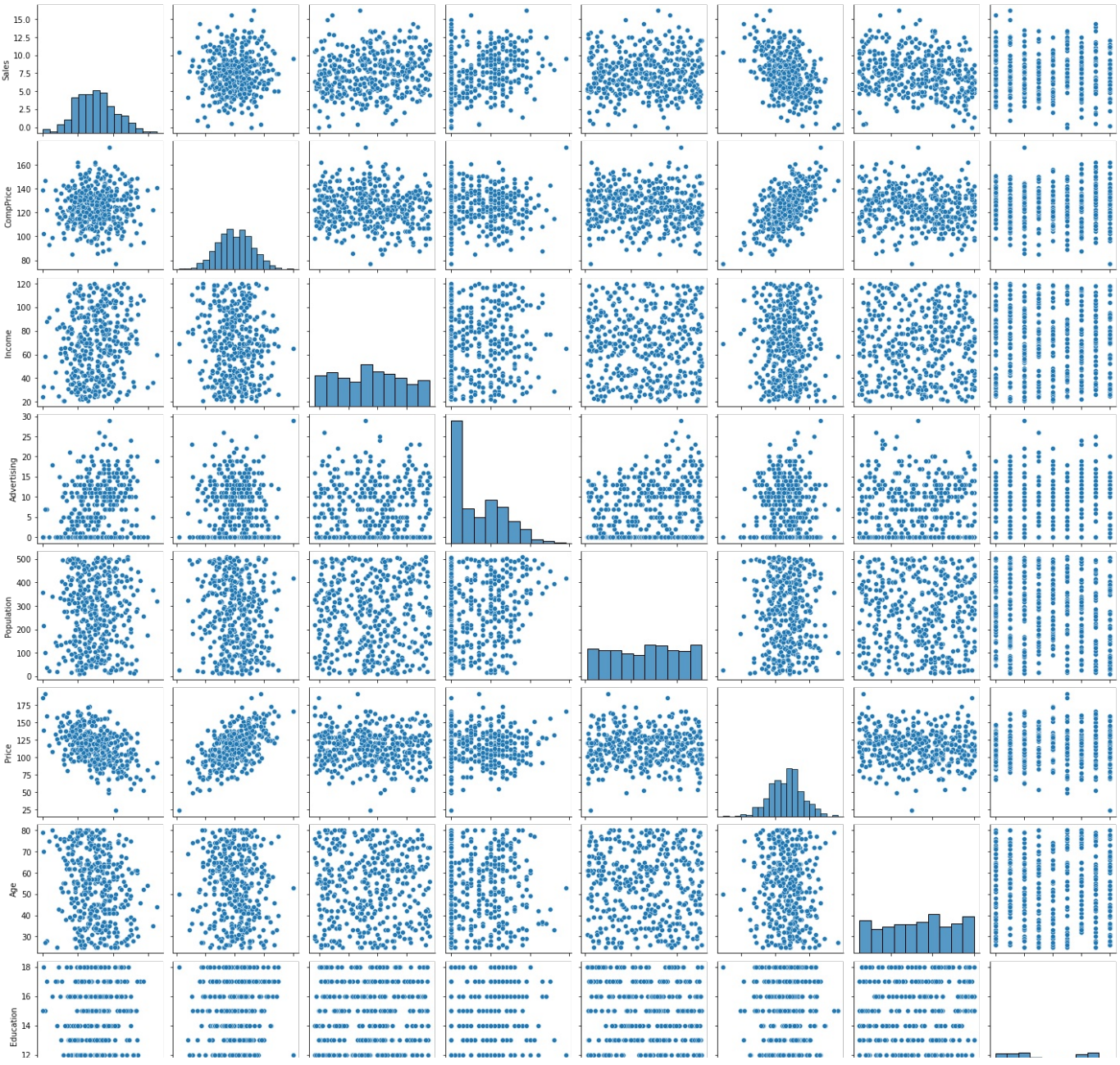
In [7]:
```
import seaborn as sns
sns.heatmap(data.isnull(),cmap='Reds')   # there are no null values
```

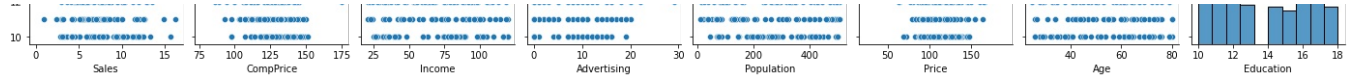
Out[7]:
```
<AxesSubplot:>
```



In [8]:
```
sns.pairplot(data)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1e5bd56d370>
```





In [9]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [10]:
```
# All other variables are independent

plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True)
```

Out[10]:
<AxesSubplot:>

In [11]:
```
#EDA
encoder = ce.OrdinalEncoder(cols=["ShelveLoc", "Urban", "US"])
sales = encoder.fit_transform(data)
```

In [12]:
```
sale_val = []
for value in data['Sales']:
    if value <= 7.49:
        sale_val.append("low")
    else:
        sale_val.append("high")

sales["sale_val"]= sale_val
```

In [13]:
```
sales.head()
```

Out[13]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	sale_val
0	9.50	138	73	11	276	120	1	42	17	1	1	high
1	11.22	111	48	16	260	83	2	65	10	1	1	high
2	10.06	113	35	10	269	80	3	59	12	1	1	high
3	7.40	117	100	4	466	97	3	55	14	1	1	low
4	4.15	141	64	3	340	128	1	38	13	1	2	low

In [14]:
```
#Train test and split
x = sales.drop(['sale_val', 'Sales'],axis=1)
```

```
y = sales['sale_val']
```

In [15]:
```
x
```

Out[15]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	138	73	11	276	120	1	42	17	1	1
1	111	48	16	260	83	2	65	10	1	1
2	113	35	10	269	80	3	59	12	1	1
3	117	100	4	466	97	3	55	14	1	1
4	141	64	3	340	128	1	38	13	1	2
...	...	...	...	...	...	...	...	...	...	...
395	138	108	17	203	128	2	33	14	1	1
396	139	23	3	37	120	3	55	11	2	1
397	162	26	12	368	159	3	40	18	1	1
398	100	79	7	284	95	1	50	12	1	1
399	134	37	0	27	120	2	49	16	1	1

400 rows × 10 columns

In [16]:
```
y
```

Out[16]:
```
0      high
1      high
2      high
3       low
4       low
       ...
395    high
396     low
397     low
398     low
399    high
Name: sale_val, Length: 400, dtype: object
```

Random Forest Classification

In [17]:
```
num_trees = 100
max_features = 4
kfold = KFold(n_splits=20 ,shuffle=True)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, x, y, cv=kfold)
print(results.mean()*100)
```

```
81.49999999999999
```

ensemble techniques

In [18]:
```
from sklearn.ensemble import BaggingClassifier
```

BAGGING

BAGGING DECISION TREE FOR CLASSIFIER

In [21]:
```
seed = 7
kfold = KFold(n_splits=20)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator = cart, n_estimators=num_trees, random_state = seed)
results = cross_val_score(model,x,y,cv = kfold)
print(results.mean())
```

```
0.8225
```

BOOSTING

In [22]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [24]:
```
num_trees = 200
seed = 7
kfold = KFold(n_splits=20)
model =  AdaBoostClassifier (n_estimators = num_trees, random_state= seed)
results = cross_val_score(model,x,y, cv=kfold)
print(results.mean())
```

```
0.8225
```

Stacking

In [25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

In [26]:
```
estimators = []
model3 = LogisticRegression(max_iter=500)
estimators.append(('logistic', model3))
model4 = DecisionTreeClassifier()
estimators.append(('cart', model4))
model5 = SVC()
estimators.append(('svm', model5))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results_stack = cross_val_score(ensemble, x, y, cv=kfold)
print(results_stack.mean()*100)
```

```
78.25
```

Conclusion: Bagging technique has a great accuracy 82.00%

In [ ]:
```

```