In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")
```

In [2]:
```python
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 [3]:
```python
data=pd.read_csv("Fraud_check.csv")
data.head()
```

Out[3]:

| | Undergrad | Marital.Status | Taxable.Income | City.Population | Work.Experience | Urban |
|---|---|---|---|---|---|---|
| **0** | NO | Single | 68833 | 50047 | 10 | YES |
| **1** | YES | Divorced | 33700 | 134075 | 18 | YES |
| **2** | NO | Married | 36925 | 160205 | 30 | YES |
| **3** | YES | Single | 50190 | 193264 | 15 | YES |
| **4** | NO | Married | 81002 | 27533 | 28 | NO |

In [4]:
```python
data.shape
```

Out[4]:
```
(600, 6)
```

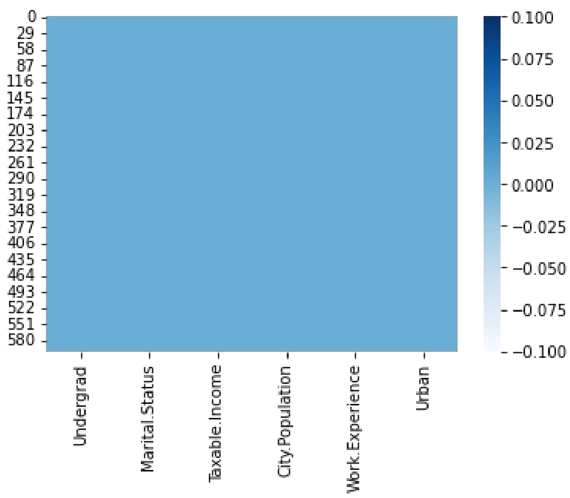
In [5]:
```python
data.describe().round(2).style.background_gradient(cmap = 'Oranges')
```

Out[5]:

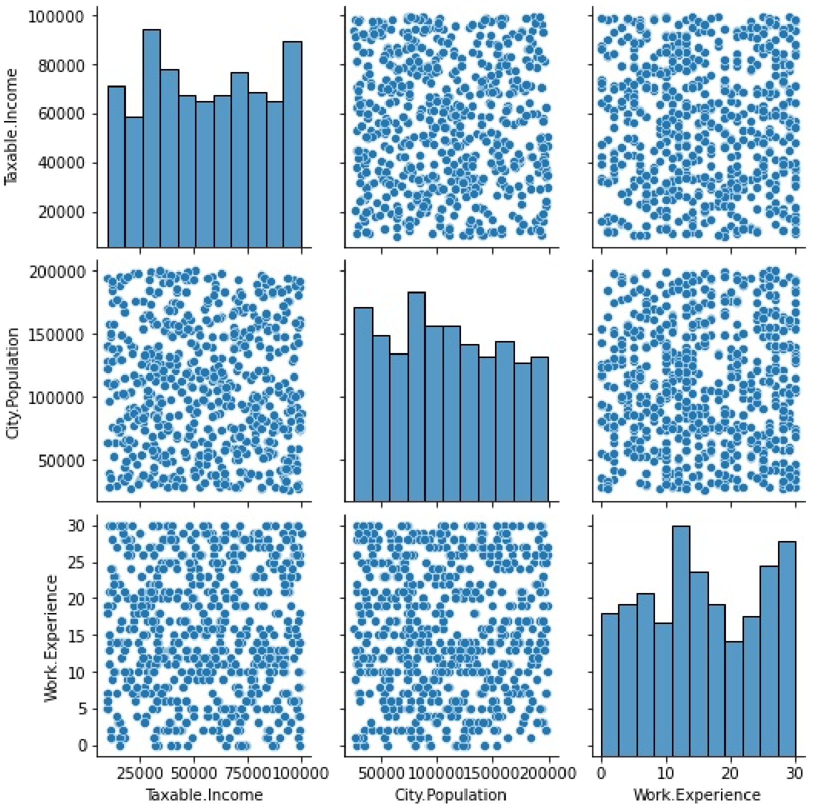
| | Taxable.Income | City.Population | Work.Experience |
|---|---|---|---|
| **count** | 600.000000 | 600.000000 | 600.000000 |
| **mean** | 55208.380000 | 108747.370000 | 15.560000 |
| **std** | 26204.830000 | 49850.080000 | 8.840000 |
| **min** | 10003.000000 | 25779.000000 | 0.000000 |
| **25%** | 32871.500000 | 66966.750000 | 8.000000 |
| **50%** | 55074.500000 | 106493.500000 | 15.000000 |
| **75%** | 78611.750000 | 150114.250000 | 24.000000 |
| **max** | 99619.000000 | 199778.000000 | 30.000000 |

In [6]:
```python
sns.heatmap(data.isnull(),cmap='Blues') # there are no null values
```

Out[6]:
```
<AxesSubplot:>
```

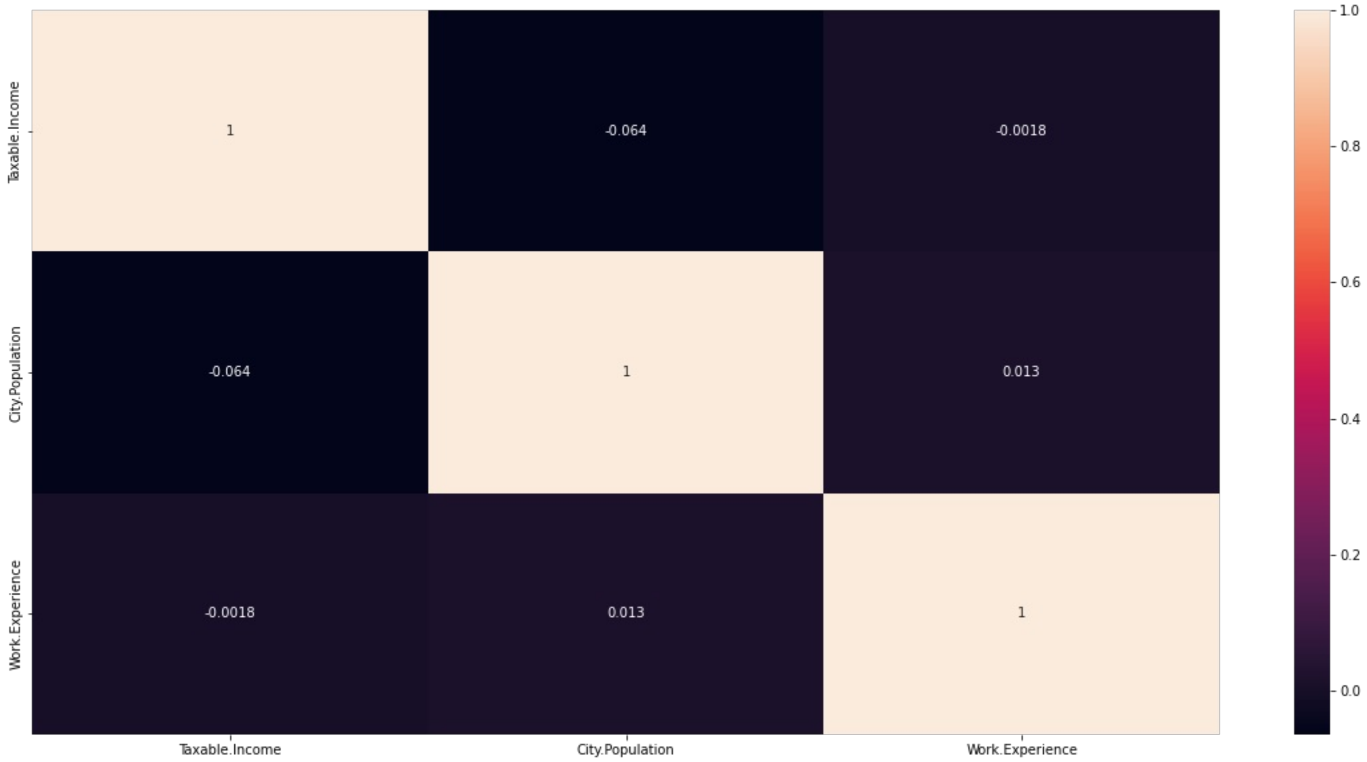


In [7]:
```python
sns.pairplot(data)
```
Out[7]: <seaborn.axisgrid.PairGrid at 0x21fa22a57c0>




In [8]:
```python
# All other variables are independent

plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True)
```
Out[8]: <AxesSubplot:>




In [9]:
```python
from sklearn import preprocessing
```

In [10]:

```
label_encode = preprocessing.LabelEncoder()
data['Undergrad'] = label_encode.fit_transform(data['Undergrad'])
data['Marital.Status'] = label_encode.fit_transform(data['Marital.Status'])
data['Urban'] = label_encode.fit_transform(data['Urban'])
```

In [11]:

```
data
```

Out[11]:

| | Undergrad | Marital.Status | Taxable.Income | City.Population | Work.Experience | Urban |
|---|---|---|---|---|---|---|
| **0** | 0 | 2 | 68833 | 50047 | 10 | 1 |
| **1** | 1 | 0 | 33700 | 134075 | 18 | 1 |
| **2** | 0 | 1 | 36925 | 160205 | 30 | 1 |
| **3** | 1 | 2 | 50190 | 193264 | 15 | 1 |
| **4** | 0 | 1 | 81002 | 27533 | 28 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **595** | 1 | 0 | 76340 | 39492 | 7 | 1 |
| **596** | 1 | 0 | 69967 | 55369 | 2 | 1 |
| **597** | 0 | 0 | 47334 | 154058 | 0 | 1 |
| **598** | 1 | 1 | 98592 | 180083 | 17 | 0 |
| **599** | 0 | 0 | 96519 | 158137 | 16 | 0 |

600 rows × 6 columns

In [12]:

```
data['Status'] = data['Taxable.Income'].apply(lambda Income: 'Risky' if Income <= 30000 else 'Good')
```

In [13]:

```
data
```

Out[13]:

| | Undergrad | Marital.Status | Taxable.Income | City.Population | Work.Experience | Urban | Status |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 2 | 68833 | 50047 | 10 | 1 | Good |
| **1** | 1 | 0 | 33700 | 134075 | 18 | 1 | Good |
| **2** | 0 | 1 | 36925 | 160205 | 30 | 1 | Good |
| **3** | 1 | 2 | 50190 | 193264 | 15 | 1 | Good |
| **4** | 0 | 1 | 81002 | 27533 | 28 | 0 | Good |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **595** | 1 | 0 | 76340 | 39492 | 7 | 1 | Good |
| **596** | 1 | 0 | 69967 | 55369 | 2 | 1 | Good |
| **597** | 0 | 0 | 47334 | 154058 | 0 | 1 | Good |
| **598** | 1 | 1 | 98592 | 180083 | 17 | 0 | Good |
| **599** | 0 | 0 | 96519 | 158137 | 16 | 0 | Good |

600 rows × 7 columns

In [14]:

```
x = data.iloc[:,0:5]
y = data['Status']
```

In [15]:

```
x
```

Out[15]:

| | Undergrad | Marital.Status | Taxable.Income | City.Population | Work.Experience |
|---|---|---|---|---|---|
| **0** | 0 | 2 | 68833 | 50047 | 10 |
| **1** | 1 | 0 | 33700 | 134075 | 18 |
| **2** | 0 | 1 | 36925 | 160205 | 30 |
| **3** | 1 | 2 | 50190 | 193264 | 15 |
| **4** | 0 | 1 | 81002 | 27533 | 28 |
| **...** | ... | ... | ... | ... | ... |
| **595** | 1 | 0 | 76340 | 39492 | 7 |
| **596** | 1 | 0 | 69967 | 55369 | 2 |
| **597** | 0 | 0 | 47334 | 154058 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| **598** | 1 | 1 | 98592 | 180083 | 17 |
| **599** | 0 | 0 | 96519 | 158137 | 16 |

600 rows × 5 columns

In [16]:

```
y
```

Out[16]:

```
0      Good
1      Good
2      Good
3      Good
4      Good
       ...
595    Good
596    Good
597    Good
598    Good
599    Good
Name: Status, Length: 600, dtype: object
```

In [17]:

```
data['Status'].unique()
```

Out[17]:

```
array(['Good', 'Risky'], dtype=object)
```

In [18]:

```
# split the datax_
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)
```

In [19]:

```
(x_train.shape),(x_test.shape),(y_train.shape),(y_test.shape)
```

Out[19]:

```
((420, 5), (180, 5), (420,), (180,))
```

In [20]:

```
x_train
```

Out[20]:

| | Undergrad | Marital.Status | Taxable.Income | City.Population | Work.Experience |
|---|---|---|---|---|---|
| **119** | 0 | 0 | 97318 | 47202 | 30 |
| **482** | 0 | 2 | 32786 | 125771 | 12 |
| **470** | 1 | 1 | 52663 | 148686 | 26 |
| **173** | 1 | 2 | 84835 | 105110 | 16 |
| **196** | 1 | 1 | 10933 | 28410 | 21 |
| **...** | ... | ... | ... | ... | ... |
| **96** | 0 | 2 | 22258 | 63622 | 17 |
| **277** | 0 | 0 | 63710 | 117364 | 11 |
| **414** | 0 | 0 | 97980 | 27300 | 1 |
| **379** | 0 | 2 | 26101 | 112774 | 13 |
| **276** | 0 | 2 | 62426 | 44251 | 17 |

420 rows × 5 columns

In [21]:

```
y_test
```

Out[21]:

```
222     Good
30      Good
389     Good
292    Risky
21     Risky
        ...
41      Good
556     Good
456     Good
387     Good
245     Good
```

```
Name: Status, Length: 180, dtype: object
```

# Random Forest Classification

In [22]:
```
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)
```

```
99.83333333333334
```

# ensemble technique

In [23]:
```
from sklearn.ensemble import BaggingClassifier
```

# BAGGING

BAGGING DECISION TREE FOR CLASSIFIER

In [24]:
```
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())
```

```
0.9983333333333334
```

# BOOSTING

In [25]:
```
from sklearn.ensemble import AdaBoostClassifier
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())
```

```
0.9983333333333334
```

# Stacking

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

In [27]:
```
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)
```

```
98.66666666666667
```

Conclusion: Bagging & Boosting & Stacking technique has a great accuracy 98.00%

In [ ]:

```

```