SVM_Company

Prepare a classification model using SVM for salary data

Data Description:

age -- age of a person

workclass -- A work class is a grouping of work

education -- Education of an individuals

maritalstatus -- Marital status of an individulas

occupation -- occupation of an individuals

relationship --

race -- Race of an Individual

sex -- Gender of an Individual

capitalgain -- profit received from the sale of an investment

capitalloss -- A decrease in the value of a capital asset

hoursperweek -- number of hours work per week

native -- Native of an individual

Salary -- salary of an individual

Importing the libraries

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

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split,cross_val_scor

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, confusion_matrix
```

Importing the Dataset

In [2]:
```
Train = pd.read_csv("/Users/chethantumkur/Desktop/Data Science/ASSS
Train
```

Out[2]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	ra
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Wh
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Wh
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Wh
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Bla
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Bla
...	...	...	...	...	...	...	...	
30156	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	Wh
30157	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	Wh
30158	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	Wh
30159	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	Wh
30160	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	Wh

30161 rows × 14 columns

In [3]:
```
Test = pd.read_csv("/Users/chethantumkur/Desktop/Data Science/ASSSI
Test
```

Out[3]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	E
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	V
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	V
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	E
4	34	Private	10th	6	Never-married	Other-service	Not-in-family	V
...	...	...	...	...	...	...	...	
15055	33	Private	Bachelors	13	Never-married	Prof-specialty	Own-child	V
15056	39	Private	Bachelors	13	Divorced	Prof-specialty	Not-in-family	V
15057	38	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	V
15058	44	Private	Bachelors	13	Divorced	Adm-clerical	Own-child	A Isla
15059	35	Self-emp-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	V

15060 rows × 14 columns

In [4]:
```
Train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30161 entries, 0 to 30160
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            30161 non-null  int64 
 1   workclass      30161 non-null  object
 2   education      30161 non-null  object
 3   educationno    30161 non-null  int64 
 4   maritalstatus  30161 non-null  object
 5   occupation     30161 non-null  object
 6   relationship   30161 non-null  object
 7   race           30161 non-null  object
 8   sex            30161 non-null  object
 9   capitalgain    30161 non-null  int64 
 10  capitalloss    30161 non-null  int64 
 11  hoursperweek   30161 non-null  int64 
 12  native         30161 non-null  object
 13  Salary         30161 non-null  object
dtypes: int64(5), object(9)
memory usage: 3.2+ MB
```

In [5]:
```
Train.describe()
```

Out[5]:

	age	educationno	capitalgain	capitalloss	hoursperweek
count	30161.000000	30161.000000	30161.000000	30161.000000	30161.000000
mean	38.438115	10.121316	1092.044064	88.302311	40.931269
std	13.134830	2.550037	7406.466611	404.121321	11.980182
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	47.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

In [6]:
```
Test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15060 entries, 0 to 15059
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            15060 non-null  int64 
 1   workclass      15060 non-null  object
 2   education      15060 non-null  object
 3   educationno    15060 non-null  int64 
 4   maritalstatus  15060 non-null  object
 5   occupation     15060 non-null  object
 6   relationship   15060 non-null  object
 7   race           15060 non-null  object
 8   sex            15060 non-null  object
 9   capitalgain    15060 non-null  int64 
 10  capitalloss    15060 non-null  int64 
 11  hoursperweek   15060 non-null  int64 
 12  native         15060 non-null  object
 13  Salary         15060 non-null  object
dtypes: int64(5), object(9)
memory usage: 1.6+ MB
```

In [7]:
```
Test.describe()
```

Out[7]:

	age	educationno	capitalgain	capitalloss	hoursperweek
count	15060.000000	15060.000000	15060.000000	15060.000000	15060.000000
mean	38.768327	10.112749	1120.301594	89.041899	40.951594
std	13.380676	2.558727	7703.181842	406.283245	12.062831
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	3770.000000	99.000000

In [8]:
```
Train[Train.isnull().any(axis=1)]
```

Out[8]:

age	workclass	education	educationno	maritalstatus	occupation	relationship	race	se

In [9]:
```
Train.isnull().sum()
## shows the total no of nan value in each columns
## there is no nan values in the Train Data set
```

Out[9]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [10]:
```
Test[Test.isnull().any(axis=1)].shape
## shows the total no of nan value in each columns
## there is no nan values in the Train Data set
```

Out[10]:
```
(0, 14)
```

In [11]:
```
Train['Salary'].value_counts()
```

Out[11]:
```
 <=50K    22653
 >50K      7508
Name: Salary, dtype: int64
```

In [12]:
```
Test['Salary'].value_counts()
```

Out[12]:
```
 <=50K    11360
 >50K      3700
Name: Salary, dtype: int64
```

In [13]:
```
pd.crosstab(Train['occupation'],Train['Salary'])
```

Out[13]:

Salary occupation	<=50K	>50K
Adm-clerical	3223	498
Armed-Forces	8	1
Craft-repair	3122	908
Exec-managerial	2055	1937
Farming-fishing	874	115
Handlers-cleaners	1267	83
Machine-op-inspct	1720	245
Other-service	3080	132
Priv-house-serv	142	1
Prof-specialty	2227	1811
Protective-serv	434	210
Sales	2614	970
Tech-support	634	278
Transport-moving	1253	319

In [14]:
```
pd.crosstab(Train['workclass'],Train['Salary'])
```

Out[14]:

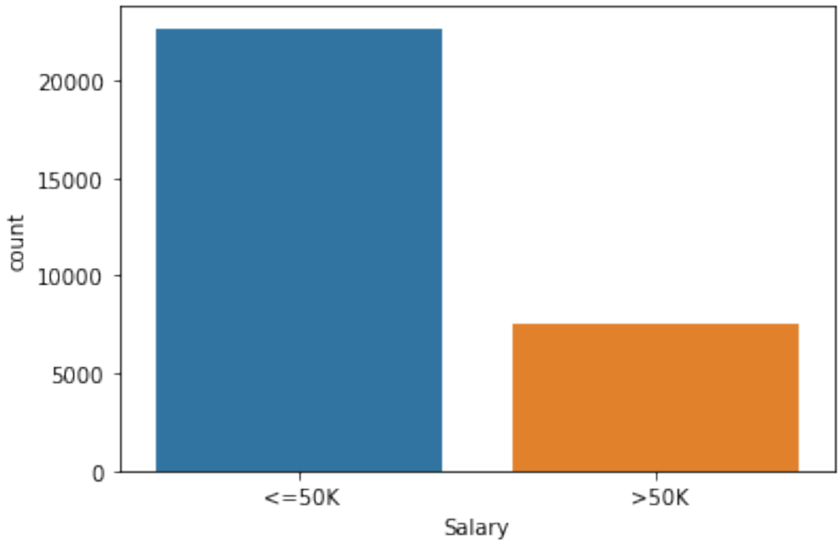
Salary workclass	<=50K	>50K
Federal-gov	578	365
Local-gov	1458	609
Private	17409	4876
Self-emp-inc	474	600
Self-emp-not-inc	1785	714
State-gov	935	344
Without-pay	14	0

In [15]:
```
pd.crosstab(Train['workclass'],Train['occupation'])
```

Out[15]:

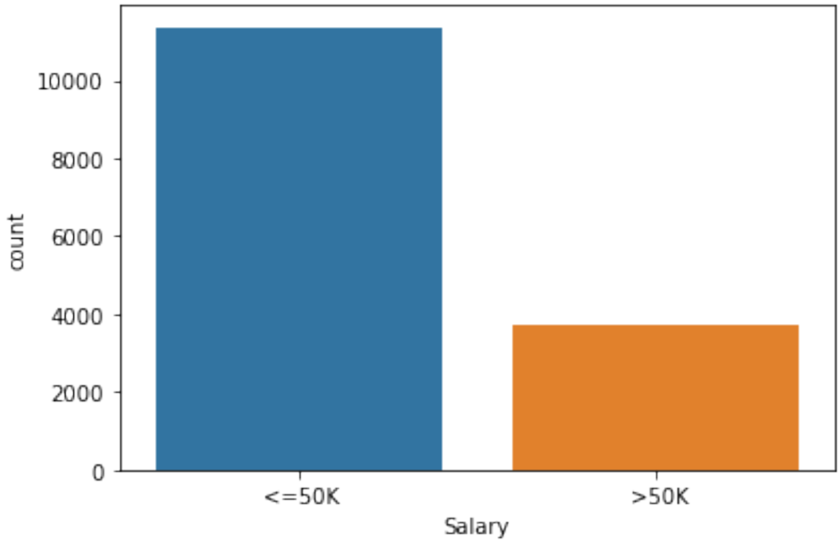
occupation / workclass	Adm-clerical	Armed-Forces	Craft-repair	Exec-managerial	Farming-fishing	Handlers-cleaners	Machine-op-inspct	Other-service
Federal-gov	316	9	63	179	8	22	14	34
Local-gov	281	0	143	212	29	46	11	189
Private	2793	0	3146	2647	450	1255	1881	2665
Self-emp-inc	28	0	99	385	51	2	10	27
Self-emp-not-inc	49	0	523	383	430	15	35	173
State-gov	251	0	55	186	15	9	13	123
Without-pay	3	0	1	0	6	1	1	1

In [16]:
```
sns.countplot(x='Salary',data= Train)
plt.xlabel('Salary')
plt.ylabel('count')
plt.show()
Train['Salary'].value_counts()
```



Out[16]:
```
 <=50K    22653
 >50K      7508
Name: Salary, dtype: int64
```

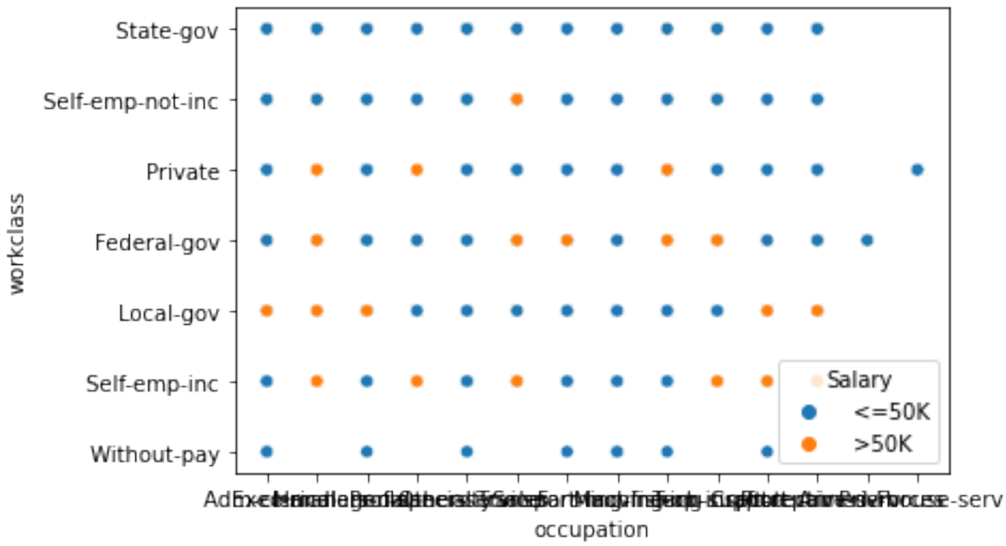
In [17]:
```
sns.countplot(x='Salary',data= Test)
plt.xlabel('Salary')
plt.ylabel('count')
plt.show()
Test['Salary'].value_counts()
```



Out[17]:
```
 <=50K    11360
 >50K      3700
Name: Salary, dtype: int64
```

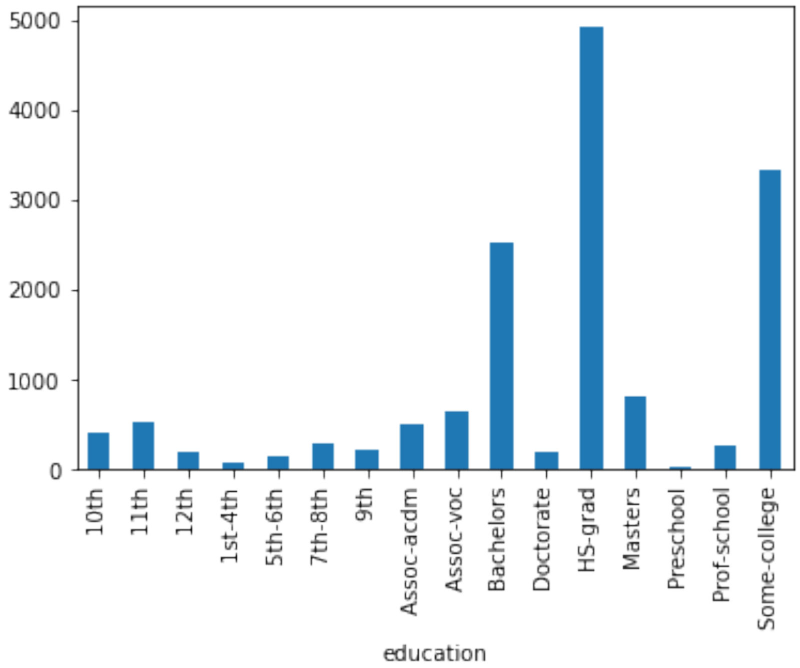
In [18]:
```
sns.scatterplot(Train['occupation'],Train['workclass'],hue=Train['S
```

Out[18]:
```
<AxesSubplot:xlabel='occupation', ylabel='workclass'>
```



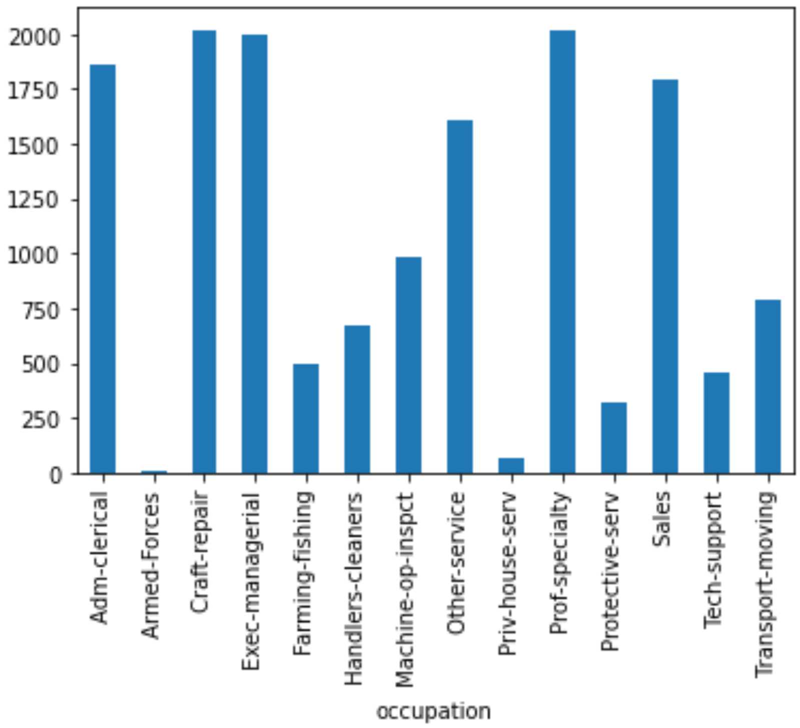
In [19]:
```
pd.crosstab(Train['Salary'],Train['education']).mean().plot(kind='b
```

Out[19]: <AxesSubplot:xlabel='education'>



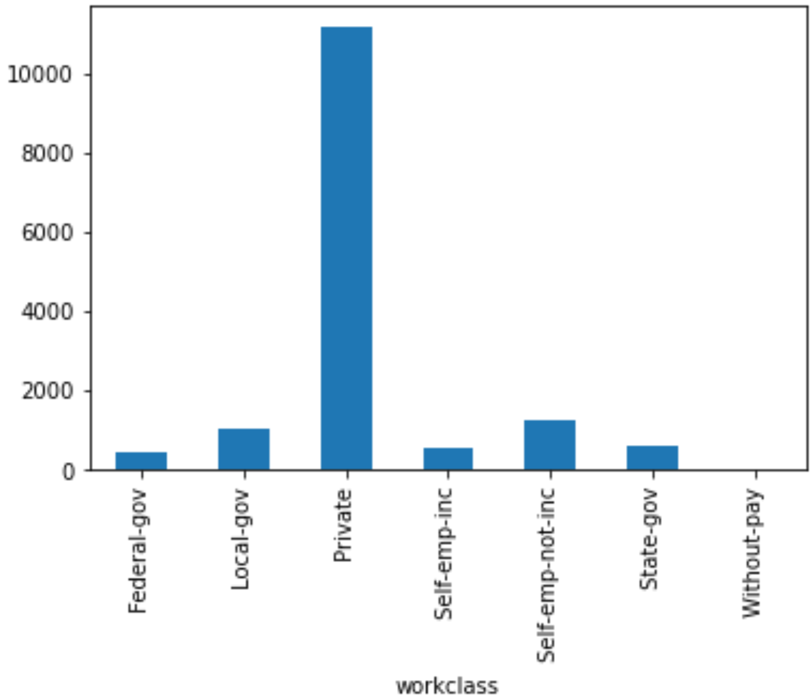
In [20]:
```
pd.crosstab(Train['Salary'],Train['occupation']).mean().plot(kind='
```

Out[20]: <AxesSubplot:xlabel='occupation'>



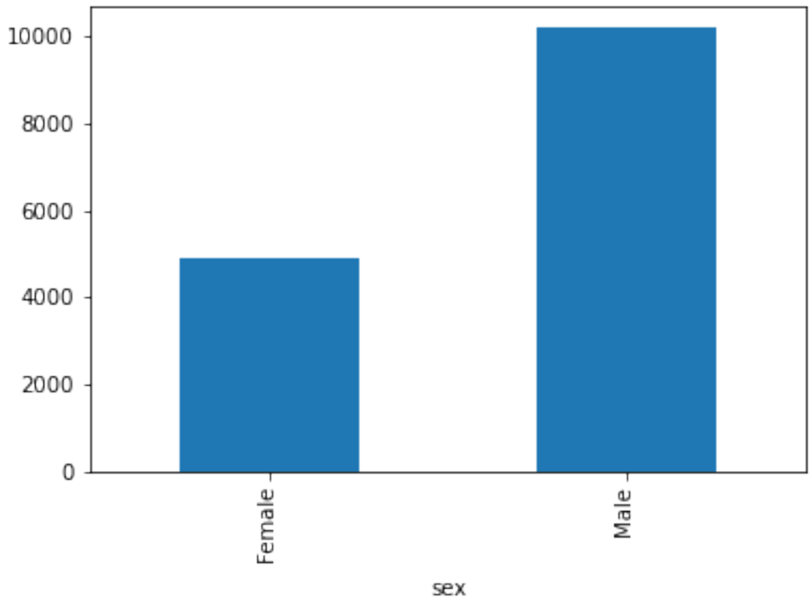
In [21]:
```
pd.crosstab(Train['Salary'],Train['workclass']).mean().plot(kind='b
```

Out[21]: <AxesSubplot:xlabel='workclass'>



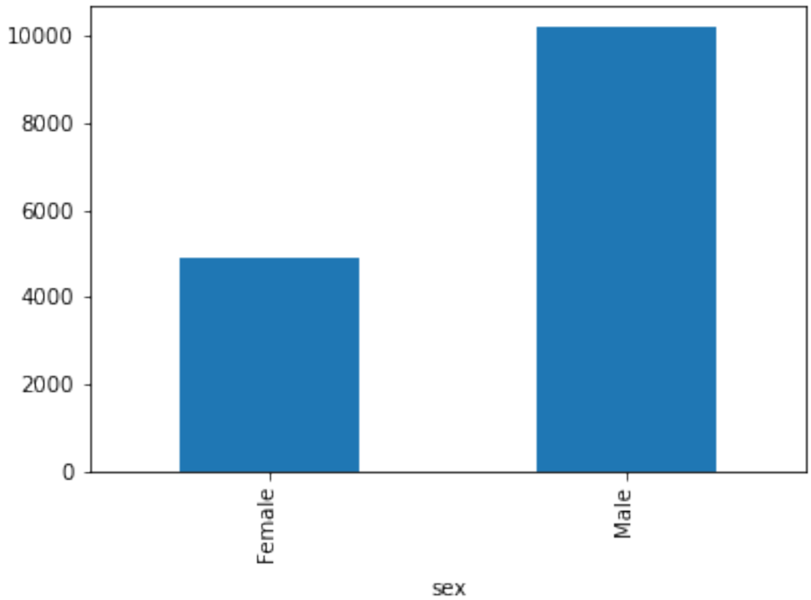
In [22]:
```
pd.crosstab(Train['Salary'],Train['sex']).mean().plot(kind='bar')
```

Out[22]: <AxesSubplot:xlabel='sex'>



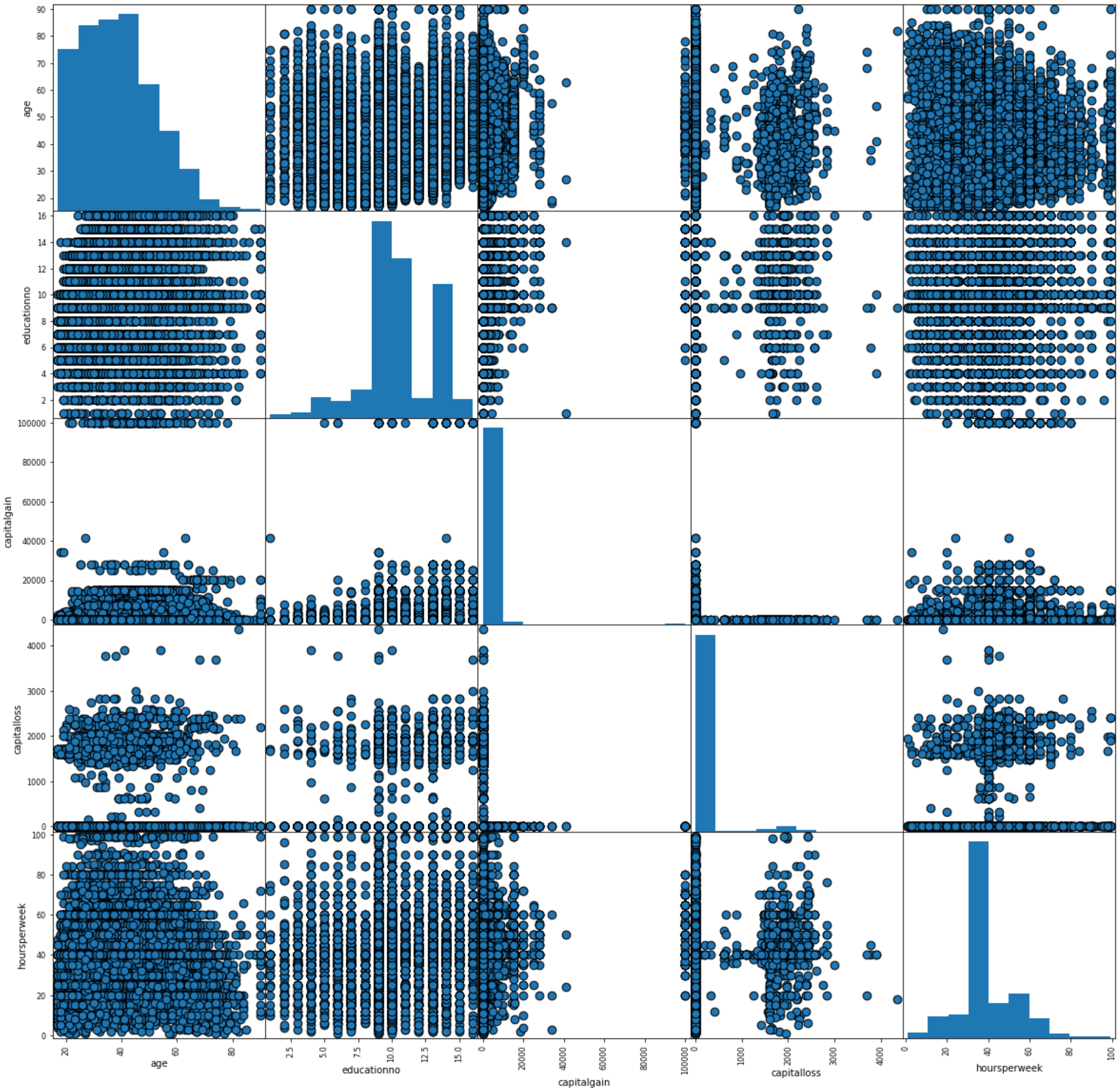
In [23]:
```
pd.crosstab(Train['Salary'],Train['sex']).mean().plot(kind='bar')
```

Out[23]:
```
<AxesSubplot:xlabel='sex'>
```



In [24]:

```
# scatter matrix to observe relationship between every colomn attri
pd.plotting.scatter_matrix(Train,
                                       figsize= [20,20],
                                       diagonal='hist',
                                       alpha=1,
                                       s = 300,
                                       marker = '.',
                                       edgecolor= "black")
plt.show()
```



In [25]:

```
string_columns = ["workclass","education","maritalstatus","occupati
```

In [26]:

```
##Preprocessing the data. As, there are categorical variables
number = LabelEncoder()
for i in string_columns:
        Train[i]= number.fit_transform(Train[i])
        Test[i]=number.fit_transform(Test[i])
```

In [27]:
```
Test
```

Out[27]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	rac
0	25	2	1	7	4	6	3	
1	38	2	11	9	2	4	0	
2	28	1	7	12	2	10	0	
3	44	2	15	10	2	6	0	
4	34	2	0	6	4	7	1	
...	...	...	...	...	...	...	...	
15055	33	2	9	13	4	9	3	
15056	39	2	9	13	0	9	1	
15057	38	2	9	13	2	9	0	
15058	44	2	9	13	0	0	3	
15059	35	3	9	13	2	3	0	

15060 rows × 14 columns

In [28]:
```
##Capturing the column names which can help in futher process
colnames = Train.columns
colnames
```

Out[28]:
```
Index(['age', 'workclass', 'education', 'educationno', 'maritalsta
tus',
       'occupation', 'relationship', 'race', 'sex', 'capitalgain',
       'capitalloss', 'hoursperweek', 'native', 'Salary'],
      dtype='object')
```

In [29]:
```
len(colnames)
```

Out[29]: 14

In [30]:
```
Train
```

Out[30]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	rac
0	39	5	9	13	4	0	1	
1	50	4	9	13	2	3	0	
2	38	2	11	9	0	5	1	
3	53	2	1	7	2	5	0	
4	28	2	9	13	2	9	5	
...	...	...	...	...	...	...	...	
30156	27	2	7	12	2	12	5	
30157	40	2	11	9	2	6	0	
30158	58	2	11	9	6	0	4	
30159	22	2	11	9	4	0	3	
30160	52	3	11	9	2	3	5	

30161 rows × 14 columns

In [31]:
```
Test
```

Out[31]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	rac
0	25	2	1	7	4	6	3	
1	38	2	11	9	2	4	0	
2	28	1	7	12	2	10	0	
3	44	2	15	10	2	6	0	
4	34	2	0	6	4	7	1	
...	...	...	...	...	...	...	...	
15055	33	2	9	13	4	9	3	
15056	39	2	9	13	0	9	1	
15057	38	2	9	13	2	9	0	
15058	44	2	9	13	0	0	3	
15059	35	3	9	13	2	3	0	

15060 rows × 14 columns

In [32]:
```
x_train = Train[colnames[0:13]]
y_train = Train[colnames[13]]
x_test = Test[colnames[0:13]]
y_test = Test[colnames[13]]
```

In [33]:
```
##Normalmization
def norm_func(i):
    x = (i-i.min())/(i.max()-i.min())
    return (x)
```

In [34]:
```
x_train = norm_func(x_train)
x_test =  norm_func(x_test)
```

SVM With GRID SEARCH

In [35]:
```
clf= SVC()
parma_grid = [{'kernel' : ["rbf"],'random_state':[40],'gamma':[0.1]
```

In [36]:
```
gsv = GridSearchCV(clf,parma_grid,cv=10)
gsv.fit(x_train,y_train)
```

Out[36]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid=[{'C': [1.0], 'gamma': [0.1], 'kernel': ['
rbf'],
                          'random_state': [40]}])
```

In [37]:
```
gsv.best_params_ , gsv.best_score_
```

Out[37]:
```
({'C': 1.0, 'gamma': 0.1, 'kernel': 'rbf', 'random_state': 40},
 0.8294490262517703)
```

In [38]:
```
clf = SVC(C= 15, gamma = 50)
clf.fit(x_train , y_train)
y_pred = clf.predict(x_test)
```

In [39]:
```
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 80.64409030544488
```

In [40]:
```
confusion_matrix(y_test, y_pred)
```

Out[40]:
```
array([[10365,   995],
       [ 1920,  1780]])
```

In [41]:
```
# kernel = linear
clf= SVC()
parma_grid = [{'kernel' : ["linear"],'random_state':[40],'gamma':[0
```

In [42]:
```
gsv = GridSearchCV(clf,parma_grid,cv=10)
gsv.fit(x_train,y_train)
```

Out[42]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid=[{'C': [1.0], 'gamma': [0.1], 'kernel': ['
linear'],
                          'random_state': [40]}])
```

In [43]:
```
gsv.best_params_ , gsv.best_score_
```

Out[43]:
```
({'C': 1.0, 'gamma': 0.1, 'kernel': 'linear', 'random_state': 40},
 0.8118431562437083)
```

In [44]:
```
clf = SVC(C= 15, gamma = 50)
clf.fit(x_train , y_train)
y_pred = clf.predict(x_test)
```

In [45]:
```
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 80.64409030544488
```

In [46]:
```
confusion_matrix(y_test, y_pred)
```

Out[46]:
```
array([[10365,   995],
       [ 1920,  1780]])
```

In [47]:
```
# kernel = poly
clf= SVC()
parma_grid = [{'kernel' : ["poly"],'random_state':[40],'gamma':[0.1
```

In [48]:
```
gsv = GridSearchCV(clf,parma_grid,cv=10)
gsv.fit(x_train,y_train)
```

Out[48]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid=[{'C': [1.0], 'gamma': [0.1], 'kernel': ['
poly'],
                          'random_state': [40]}])
```

In [49]:
```
gsv.best_params_ , gsv.best_score_
```

Out[49]:
```
({'C': 1.0, 'gamma': 0.1, 'kernel': 'poly', 'random_state': 40},
 0.819137640901382)
```

In [50]:
```
clf = SVC(C= 15, gamma = 50)
clf.fit(x_train , y_train)
y_pred = clf.predict(x_test)
```

In [51]:
```
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 80.64409030544488
```

In [52]:
```
# kernel = sigmoid
clf= SVC()
parma_grid = [{'kernel' : ["sigmoid"],'random_state':[40],'gamma':[
```

In [53]:
```
gsv = GridSearchCV(clf,parma_grid,cv=10)
gsv.fit(x_train,y_train)
```

Out[53]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid=[{'C': [1.0], 'gamma': [0.1], 'kernel': ['
sigmoid'],
                          'random_state': [40]}])
```

In [54]:
```
gsv.best_params_ , gsv.best_score_
```

Out[54]:
```
({'C': 1.0, 'gamma': 0.1, 'kernel': 'sigmoid', 'random_state': 40}
,
 0.7954645932114129)
```

In [55]:
```
clf = SVC(C= 15, gamma = 50)
clf.fit(x_train , y_train)
y_pred = clf.predict(x_test)
```

In [56]:
```
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 80.64409030544488
```

In [ ]:

SVM Model

KERNAL= LINEAR

In [58]:
```
model_linear = SVC(kernel = "linear",random_state=40,gamma=0.1,C=1.
model_linear.fit(x_train,y_train)
```

Out[58]:
```
SVC(gamma=0.1, kernel='linear', random_state=40)
```

In [59]:
```
pred_test_linear = model_linear.predict(x_test)
```

In [60]:
```
np.mean(pred_test_linear==y_test) # Accuracy = 80.98%
```

Out[60]:
```
0.8098273572377158
```

In [ ]:

KERNAL= POLYNOMIAL

In [61]:
```
# Kernel = poly
model_poly = SVC(kernel = "poly",random_state=40,gamma=0.1,C=1.0)
model_poly.fit(x_train,y_train)
pred_test_poly = model_poly.predict(x_test)
```

In [62]:
```
np.mean(pred_test_poly==y_test) # Accuracy = 82.05%
```

Out[62]:
```
0.8205179282868525
```

KERNAL= Radial Basis Function

In [63]:
```
# kernel = rbf
model_rbf = SVC(kernel = "rbf",random_state=40,gamma=0.1,C=1.0)
model_rbf.fit(x_train,y_train)
pred_test_rbf = model_rbf.predict(x_test)
```

In [64]:
```
np.mean(pred_test_rbf==y_test) # Accuracy = 82.80%
```

Out[64]:
```
0.8280876494023904
```

KERNAL= Sigmoid

In [65]:
```
#'sigmoid'
model_sig = SVC(kernel = "sigmoid",random_state=40,gamma=0.1,C=1.0)
model_sig.fit(x_train,y_train)
pred_test_sig = model_rbf.predict(x_test)
```

In [66]:
```
np.mean(pred_test_sig==y_test) #Accuracy = 82.80%
```

Out[66]:
```
0.8280876494023904
```

Conclusion:

Kernal Linear is more Accurate.

In [ ]:

```
```

In [ ]:

```
```