In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
vstable = pd.read_csv("http://facweb.cs.depaul.edu/mobasher/classes/csc478/c

vstable.shape
```

Out[2]:
```
(40, 6)
```

In [3]:
```
vstable.head(10)
```

Out[3]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre
ID						
1	M	45000	25	27	2.5	Action
2	F	54000	33	12	3.4	Drama
3	F	32000	20	42	1.6	Comedy
4	F	59000	70	16	4.2	Drama
5	M	37000	35	25	3.2	Action
6	M	18000	20	33	1.7	Action
7	F	29000	45	19	3.8	Drama
8	M	74000	25	31	2.4	Action
9	M	38000	21	18	2.1	Comedy
10	F	65000	40	21	3.3	Drama

In [4]:
```
vstable.columns
```

Out[4]:
```
Index(['Gender', 'Income', 'Age', 'Rentals', 'AvgPerVisit', 'Genre'], dtyp
```

In [5]:
```
vstable.dtypes
```

Out[5]:
```
Gender          object
Income           int64
Age              int64
Rentals          int64
AvgPerVisit    float64
Genre           object
dtype: object
```

Now we can convert columns to the appropriate type as necessary:

In [6]:
```
vstable["Income"] = vstable["Income"].astype(float)
vstable.dtypes
```

Out[6]:
```
Gender          object
Income         float64
Age              int64
Rentals          int64
AvgPerVisit    float64
Genre           object
dtype: object
```

In [7]:
```
vstable.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40 entries, 1 to 40
Data columns (total 6 columns):
Gender         40 non-null object
Income         40 non-null float64
Age            40 non-null int64
Rentals        40 non-null int64
AvgPerVisit    40 non-null float64
Genre          40 non-null object
dtypes: float64(2), int64(2), object(2)
memory usage: 2.2+ KB
```

In [8]:
```
vstable.describe()
```

Out[8]:

	Income	Age	Rentals	AvgPerVisit
count	40.000000	40.000000	40.000000	40.000000
mean	41500.000000	31.500000	26.175000	2.792500
std	22925.744123	12.752074	9.594035	0.833401
min	1000.000000	15.000000	11.000000	1.200000
25%	24750.000000	21.000000	19.000000	2.200000
50%	41000.000000	30.000000	25.000000	2.800000
75%	57500.000000	36.500000	32.250000	3.325000
max	89000.000000	70.000000	48.000000	4.600000

In [9]:
```
min_sal = vstable["Income"].min()
max_sal = vstable["Income"].max()
print(min_sal, max_sal)
```

```
1000.0 89000.0
```

In [10]:
```
vstable.describe(include="all")
```

Out[10]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre
count	40	40.000000	40.000000	40.000000	40.000000	40
unique	2	NaN	NaN	NaN	NaN	3
top	M	NaN	NaN	NaN	NaN	Action
freq	21	NaN	NaN	NaN	NaN	15
mean	NaN	41500.000000	31.500000	26.175000	2.792500	NaN
std	NaN	22925.744123	12.752074	9.594035	0.833401	NaN
min	NaN	1000.000000	15.000000	11.000000	1.200000	NaN
25%	NaN	24750.000000	21.000000	19.000000	2.200000	NaN
50%	NaN	41000.000000	30.000000	25.000000	2.800000	NaN
75%	NaN	57500.000000	36.500000	32.250000	3.325000	NaN
max	NaN	89000.000000	70.000000	48.000000	4.600000	NaN

In [11]:
```
vstable[["Income", "Age"]].describe()
```

Out[11]:

	Income	Age
count	40.000000	40.000000
mean	41500.000000	31.500000
std	22925.744123	12.752074
min	1000.000000	15.000000
25%	24750.000000	21.000000
50%	41000.000000	30.000000
75%	57500.000000	36.500000
max	89000.000000	70.000000

We can perform data transformations such as normalization by directly applying the operation to the Pandas Series:

In [12]:
```
norm_sal = (vstable["Income"] - min_sal) / (max_sal-min_sal)
norm_sal.head(10)
```

Out[12]:
```
ID
1     0.500000
2     0.602273
3     0.352273
4     0.659091
5     0.409091
6     0.193182
7     0.318182
8     0.829545
9     0.420455
10    0.727273
Name: Income, dtype: float64
```

Z-Score Standardization on Age

In [13]:
```
age_z = (vstable["Age"] - vstable["Age"].mean()) / vstable["Age"].std()
age_z.head(5)
```

Out[13]:
```
ID
1   -0.509721
2    0.117628
3   -0.901814
4    3.019117
5    0.274465
Name: Age, dtype: float64
```

New columns can be added to the dataframe as needed

In [14]:
```
vstable["Age-Std"] = age_z
vstable.head()
```

Out[14]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre	Age-Std
ID							
1	M	45000.0	25	27	2.5	Action	-0.509721
2	F	54000.0	33	12	3.4	Drama	0.117628
3	F	32000.0	20	42	1.6	Comedy	-0.901814
4	F	59000.0	70	16	4.2	Drama	3.019117
5	M	37000.0	35	25	3.2	Action	0.274465

Discretization with Panda

In [15]:

```
# Discretize variable into equal-sized buckets based on rank or based on sam

inc_bins = pd.qcut(vstable.Income, 3)
inc_bins.head(10)
```

Out[15]:

```
ID
1     (29000.0, 49000.0]
2     (49000.0, 89000.0]
3     (29000.0, 49000.0]
4     (49000.0, 89000.0]
5     (29000.0, 49000.0]
6     (999.999, 29000.0]
7     (999.999, 29000.0]
8     (49000.0, 89000.0]
9     (29000.0, 49000.0]
10    (49000.0, 89000.0]
Name: Income, dtype: category
Categories (3, interval[float64]): [(999.999, 29000.0] < (29000.0, 49000.0]
```

In [16]:

```
# We can specifiy an array of quantiles for discretization together with lab

inc_bins = pd.qcut(vstable.Income, [0, .33, .66, 1], labels=["low", "mid", '
inc_bins.head(10)
```

Out[16]:

```
ID
1      mid
2     high
3      mid
4     high
5      mid
6      low
7      low
8     high
9      mid
10    high
Name: Income, dtype: category
Categories (3, object): [low < mid < high]
```

In [17]:

```
vstable["inc-bins"] = inc_bins
vstable.head(10)
```

Out[17]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre	Age-Std	inc-bins
ID								
1	M	45000.0	25	27	2.5	Action	-0.509721	mid
2	F	54000.0	33	12	3.4	Drama	0.117628	high
3	F	32000.0	20	42	1.6	Comedy	-0.901814	mid
4	F	59000.0	70	16	4.2	Drama	3.019117	high
5	M	37000.0	35	25	3.2	Action	0.274465	mid
6	M	18000.0	20	33	1.7	Action	-0.901814	low
7	F	29000.0	45	19	3.8	Drama	1.058651	low
8	M	74000.0	25	31	2.4	Action	-0.509721	high
9	M	38000.0	21	18	2.1	Comedy	-0.823395	mid
10	F	65000.0	40	21	3.3	Drama	0.666558	high

In [18]:

```
# We can also drop columns from the dataframe

vstable.drop(columns=['Age-Std','inc-bins'], inplace=True)
vstable.head()
```

Out[18]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre
ID						
1	M	45000.0	25	27	2.5	Action
2	F	54000.0	33	12	3.4	Drama
3	F	32000.0	20	42	1.6	Comedy
4	F	59000.0	70	16	4.2	Drama
5	M	37000.0	35	25	3.2	Action

In [19]:

```
vs_numeric = vstable[["Age","Income","Rentals","AvgPerVisit"]]
vs_num_std = (vs_numeric - vs_numeric.mean()) / vs_numeric.std()
vs_num_std.head(10)
```

Out[19]:

	Age	Income	Rentals	AvgPerVisit
ID				
1	-0.509721	0.152667	0.085991	-0.350971
2	0.117628	0.545239	-1.477480	0.728941
3	-0.901814	-0.414381	1.649462	-1.430883
4	3.019117	0.763334	-1.060555	1.688862
5	0.274465	-0.196286	-0.122472	0.488960
6	-0.901814	-1.025049	0.711379	-1.310893
7	1.058651	-0.545239	-0.747860	1.208901
8	-0.509721	1.417620	0.502917	-0.470962
9	-0.823395	-0.152667	-0.852092	-0.830932
10	0.666558	1.025049	-0.539398	0.608950

In [20]:

```
zscore = lambda x: (x - x.mean()) / x.std()
vs_num_std = vs_numeric.apply(zscore)
vs_num_std.head()
```

Out[20]:

	Age	Income	Rentals	AvgPerVisit
ID				
1	-0.509721	0.152667	0.085991	-0.350971
2	0.117628	0.545239	-1.477480	0.728941
3	-0.901814	-0.414381	1.649462	-1.430883
4	3.019117	0.763334	-1.060555	1.688862
5	0.274465	-0.196286	-0.122472	0.488960

In [21]:
```
# Instead of separating the numeric attributes, we can condition the standar

zscore = lambda x: ((x - x.mean()) / x.std()) if (x.dtypes==np.float64 or x.
vs_std = vstable.copy()
vs_std.apply(zscore).head()
```

Out[21]:

ID	Gender	Income	Age	Rentals	AvgPerVisit	Genre
1	M	0.152667	-0.509721	0.085991	-0.350971	Action
2	F	0.545239	0.117628	-1.477480	0.728941	Drama
3	F	-0.414381	-0.901814	1.649462	-1.430883	Comedy
4	F	0.763334	3.019117	-1.060555	1.688862	Drama
5	M	-0.196286	0.274465	-0.122472	0.488960	Action

Grouping and aggregating data

In [22]:
```
vstable.groupby("Gender").mean()
```

Out[22]:

Gender	Income	Age	Rentals	AvgPerVisit
F	40631.578947	33.631579	27.684211	2.968421
M	42285.714286	29.571429	24.809524	2.633333

In [23]:
```
vstable.groupby("Genre").mean()
```

Out[23]:

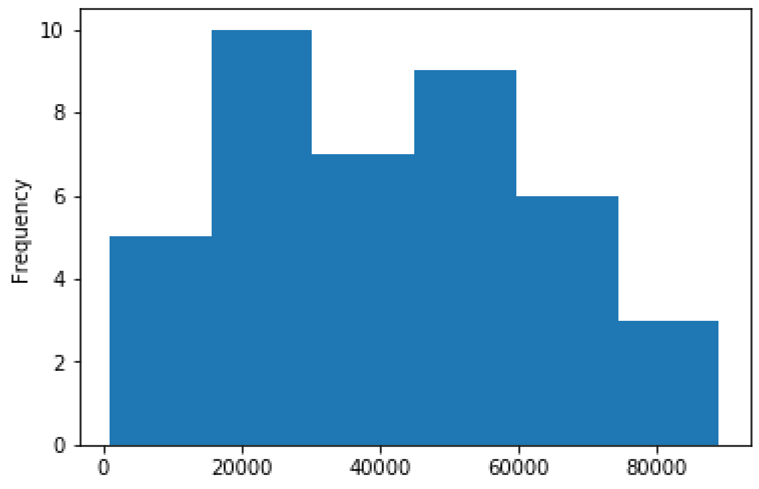
Genre	Income	Age	Rentals	AvgPerVisit
Action	32666.666667	24.066667	29.933333	2.466667
Comedy	45000.000000	31.916667	25.666667	2.641667
Drama	48461.538462	39.692308	22.307692	3.307692

In [24]:
```
vstable.groupby("Genre").describe().T
```

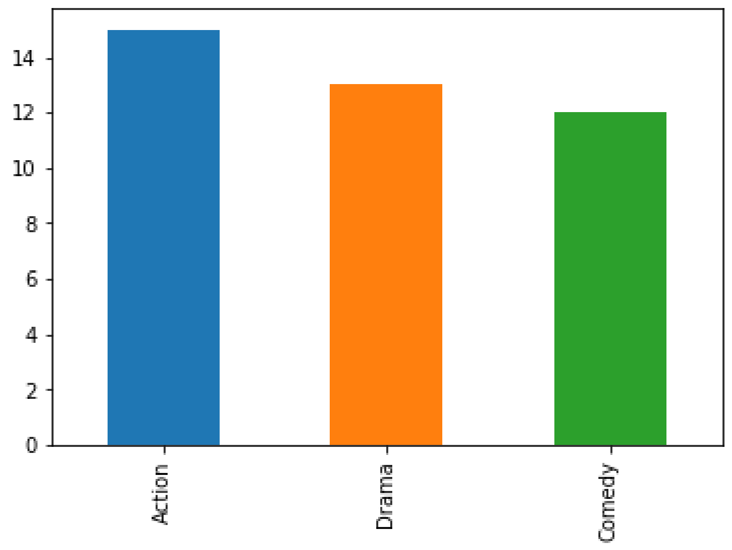
Out[24]:

	Genre	Action	Comedy	Drama
Age	**count**	15.000000	12.000000	13.000000
	mean	24.066667	31.916667	39.692308
	std	6.374802	14.650215	11.933040
	min	16.000000	15.000000	22.000000
	25%	19.000000	20.750000	33.000000
	50%	25.000000	27.500000	36.000000
	75%	27.000000	46.000000	45.000000
	max	35.000000	56.000000	70.000000
AvgPerVisit	**count**	15.000000	12.000000	13.000000
	mean	2.466667	2.641667	3.307692
	std	0.776132	0.967150	0.504086
	min	1.400000	1.200000	2.300000
	25%	1.950000	1.975000	3.100000
	50%	2.400000	2.600000	3.300000
	75%	2.800000	3.300000	3.600000
	max	4.600000	4.100000	4.200000
Income	**count**	15.000000	12.000000	13.000000
	mean	32666.666667	45000.000000	48461.538462
	std	21562.754484	29073.574381	15119.608596
	min	6000.000000	1000.000000	25000.000000
	25%	17000.000000	27750.000000	41000.000000
	50%	26000.000000	43500.000000	47000.000000
	75%	43000.000000	68000.000000	59000.000000
	max	74000.000000	89000.000000	79000.000000
Rentals	**count**	15.000000	12.000000	13.000000
	mean	29.933333	25.666667	22.307692
	std	7.591976	10.662878	9.672854
	min	17.000000	12.000000	11.000000
	25%	25.500000	17.250000	16.000000
	50%	29.000000	23.500000	21.000000
	75%	35.000000	34.500000	24.000000
	max	43.000000	42.000000	48.000000

In [25]:
```
vstable["Income"].plot(kind="hist", bins=6)
```
Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24fb262ba20>
```

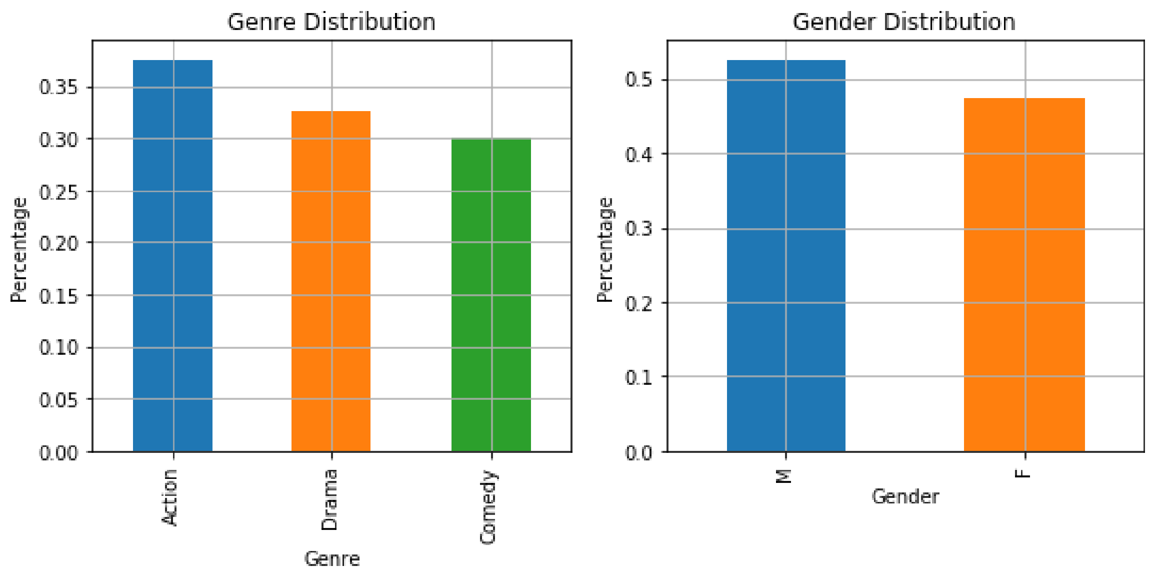


In [26]:
```
vstable["Genre"].value_counts().plot(kind='bar')
```
Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24fb26ef748>
```

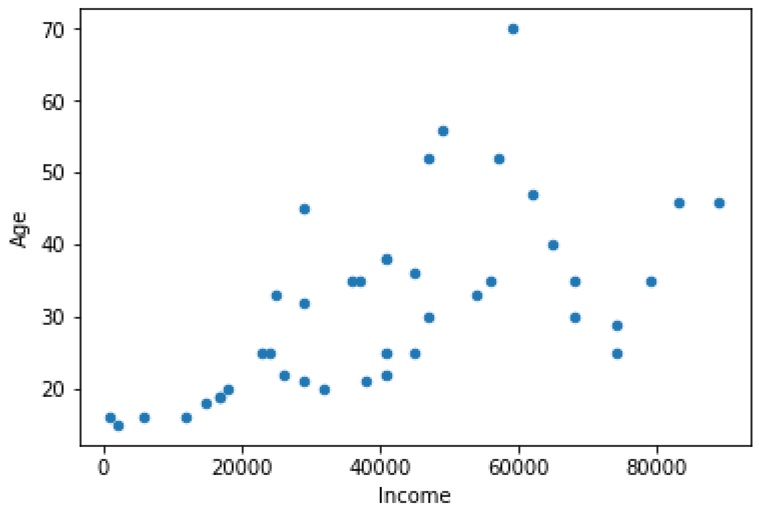


In [27]:
```
temp1 = vstable["Genre"].value_counts()/vstable["Genre"].count()
temp2 = vstable["Gender"].value_counts()/vstable["Gender"].count()
temp2

fig = plt.figure(figsize=(10,4))
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Genre')
ax1.set_ylabel('Percentage')
ax1.set_title("Genre Distribution")
temp1.plot(kind='bar', grid = True)

ax1 = fig.add_subplot(122)
ax1.set_xlabel('Gender')
ax1.set_ylabel('Percentage')
ax1.set_title("Gender Distribution")
temp2.plot(kind='bar', grid = True)
```

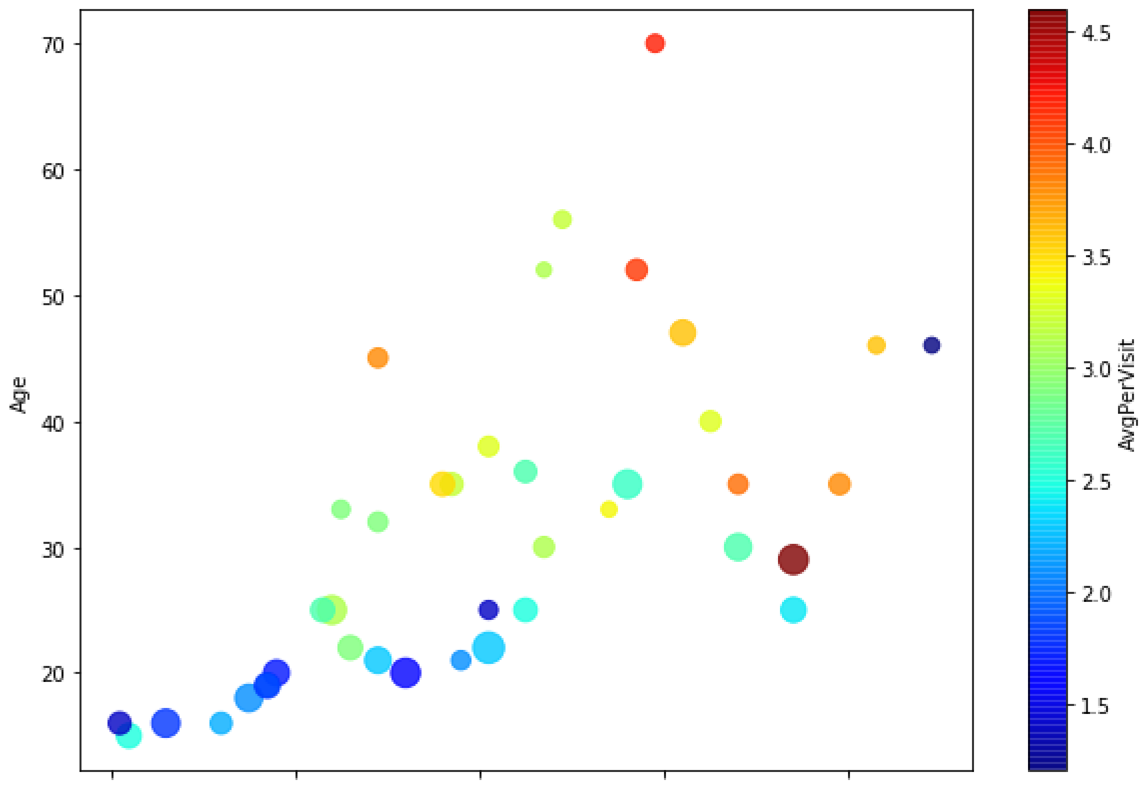
Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24fb277c860>
```



```
In [28]:  vstable.plot(x="Income", y="Age", kind="scatter")

Out[28]:  <matplotlib.axes._subplots.AxesSubplot at 0x24fb2742ba8>
```



```
In [29]:  vstable.plot(x="Income", y="Age", kind="scatter", alpha=0.8, s=vstable["Rent

Out[29]:  <matplotlib.axes._subplots.AxesSubplot at 0x24fb27cbd30>
```



In [30]:
```
vstable.groupby(["Genre","Gender"])["Gender"].count()
```

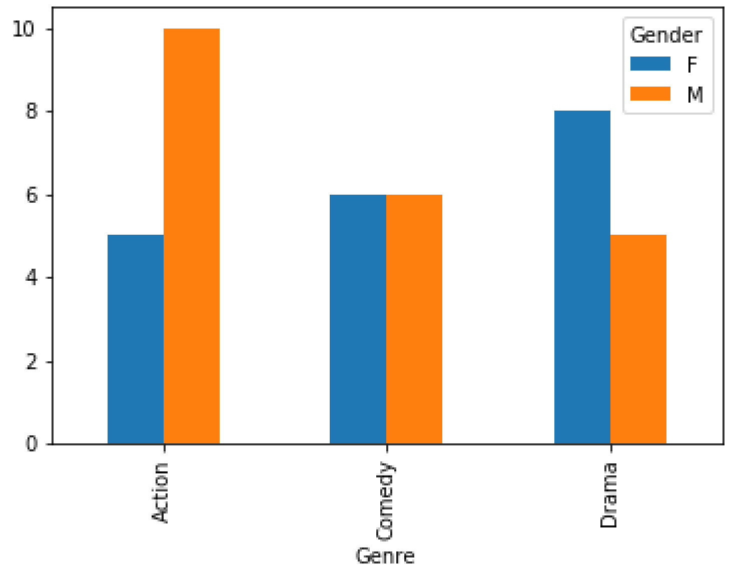
Out[30]:
```
Genre   Gender
Action  F          5
        M         10
Comedy  F          6
        M          6
Drama   F          8
        M          5
Name: Gender, dtype: int64
```

In [31]:
```
gg = pd.crosstab(vstable["Genre"], vstable["Gender"])
gg
```

Out[31]:

Gender	**F**	**M**
Genre		
Action	5	10
Comedy	6	6
Drama	8	5

In [32]:
```
plt.show(gg.plot(kind="bar"))
```

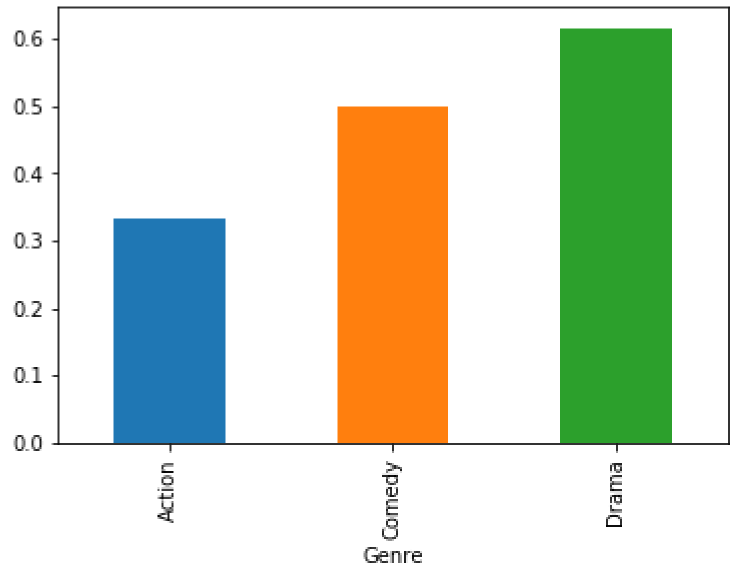


In [33]:
```
gg["percent_female"] = gg["F"]/(gg["F"]+gg["M"])
gg
```

Out[33]:

Gender	**F**	**M**	**percent_female**
Genre			
Action	5	10	0.333333
Comedy	6	6	0.500000
Drama	8	5	0.615385

In [34]:
```
plt.show(gg["percent_female"].plot(kind="bar"))
```



Suppose that we would like to find all "good cutomers", i.e., those with Rentals value of >= 30:

In [35]:
```
good_cust = vstable[vstable.Rentals>=30]
good_cust
```

Out[35]:

	Gender	Income	Age	Rentals	AvgPerVisit	Genre
ID						
3	F	32000.0	20	42	1.6	Comedy
6	M	18000.0	20	33	1.7	Action
8	M	74000.0	25	31	2.4	Action
11	F	41000.0	22	48	2.3	Drama
15	M	68000.0	30	36	2.7	Comedy
18	F	6000.0	16	39	1.8	Action
19	F	24000.0	25	41	3.1	Comedy
23	F	2000.0	15	30	2.5	Comedy
26	F	56000.0	35	40	2.6	Action
27	F	62000.0	47	32	3.6	Drama
29	F	15000.0	18	37	2.1	Action
35	M	74000.0	29	43	4.6	Action
36	F	29000.0	21	34	2.3	Comedy
40	M	17000.0	19	32	1.8	Action

In [36]:
```
print("Good Customers:")
good_cust.describe()
```

```
Good Customers:
```

Out[36]:

	Income	Age	Rentals	AvgPerVisit
count	14.000000	14.000000	14.000000	14.000000
mean	37000.000000	24.428571	37.000000	2.507143
std	25404.421178	8.599770	5.349335	0.818502
min	2000.000000	15.000000	30.000000	1.600000
25%	17250.000000	19.250000	32.250000	1.875000
50%	30500.000000	21.500000	36.500000	2.350000
75%	60500.000000	28.000000	40.750000	2.675000
max	74000.000000	47.000000	48.000000	4.600000

In [37]:
```
print("All Customers:")
vstable.describe()
```

```
All Customers:
```

Out[37]:

	Income	Age	Rentals	AvgPerVisit
count	40.000000	40.000000	40.000000	40.000000
mean	41500.000000	31.500000	26.175000	2.792500
std	22925.744123	12.752074	9.594035	0.833401
min	1000.000000	15.000000	11.000000	1.200000
25%	24750.000000	21.000000	19.000000	2.200000
50%	41000.000000	30.000000	25.000000	2.800000
75%	57500.000000	36.500000	32.250000	3.325000
max	89000.000000	70.000000	48.000000	4.600000

Creating dummy variables and converting to standard spreadsheet format (all numeric attributes)

In [38]:
```
gender_bin = pd.get_dummies(vstable["Gender"], prefix="Gender")
gender_bin.head()
```

Out[38]:
```
    Gender_F  Gender_M
ID
 1         0         1
 2         1         0
 3         1         0
 4         1         0
 5         0         1
```

In [39]:
```
vs_ssf = pd.get_dummies(vstable)
vs_ssf.head(10)
```

Out[39]:
```
     Income  Age  Rentals  AvgPerVisit  Gender_F  Gender_M  Genre_Action  Genre_Comedy
ID
 1  45000.0   25       27          2.5         0         1             1             0
 2  54000.0   33       12          3.4         1         0             0             0
 3  32000.0   20       42          1.6         1         0             0             1
 4  59000.0   70       16          4.2         1         0             0             0
 5  37000.0   35       25          3.2         0         1             1             0
 6  18000.0   20       33          1.7         0         1             1             0
 7  29000.0   45       19          3.8         1         0             0             0
 8  74000.0   25       31          2.4         0         1             1             0
 9  38000.0   21       18          2.1         0         1             0             1
10  65000.0   40       21          3.3         1         0             0             0
```

In [40]:
```
vs_ssf.describe()
```

Out[40]:

	Income	Age	Rentals	AvgPerVisit	Gender_F	Gender_M	Genre_Action
count	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.00000
mean	41500.000000	31.500000	26.175000	2.792500	0.475000	0.525000	0.37500
std	22925.744123	12.752074	9.594035	0.833401	0.505736	0.505736	0.49029
min	1000.000000	15.000000	11.000000	1.200000	0.000000	0.000000	0.00000
25%	24750.000000	21.000000	19.000000	2.200000	0.000000	0.000000	0.00000
50%	41000.000000	30.000000	25.000000	2.800000	0.000000	1.000000	0.00000
75%	57500.000000	36.500000	32.250000	3.325000	1.000000	1.000000	1.00000
max	89000.000000	70.000000	48.000000	4.600000	1.000000	1.000000	1.00000

In [41]:
```
# Min-Max normalization performed on the full numeric data set

vs_norm = (vs_ssf - vs_ssf.min()) / (vs_ssf.max()-vs_ssf.min())
vs_norm.head(10)
```

Out[41]:

ID	Income	Age	Rentals	AvgPerVisit	Gender_F	Gender_M	Genre_Action	Genre_Co
1	0.500000	0.181818	0.432432	0.382353	0.0	1.0	1.0	
2	0.602273	0.327273	0.027027	0.647059	1.0	0.0	0.0	
3	0.352273	0.090909	0.837838	0.117647	1.0	0.0	0.0	
4	0.659091	1.000000	0.135135	0.882353	1.0	0.0	0.0	
5	0.409091	0.363636	0.378378	0.588235	0.0	1.0	1.0	
6	0.193182	0.090909	0.594595	0.147059	0.0	1.0	1.0	
7	0.318182	0.545455	0.216216	0.764706	1.0	0.0	0.0	
8	0.829545	0.181818	0.540541	0.352941	0.0	1.0	1.0	
9	0.420455	0.109091	0.189189	0.264706	0.0	1.0	0.0	
10	0.727273	0.454545	0.270270	0.617647	1.0	0.0	0.0	

In [45]:

```
# After converting to all numeric attributes, we can perform correlation anc

corr_matrix = vs_ssf.corr()
corr_matrix
```

Out[45]:

	Income	Age	Rentals	AvgPerVisit	Gender_F	Gender_M	Genre_Acti
Income	1.000000	0.613769	-0.262472	0.468565	-0.036490	0.036490	-0.3022
Age	0.613769	1.000000	-0.547113	0.629107	0.161022	-0.161022	-0.4572
Rentals	-0.262472	-0.547113	1.000000	-0.206353	0.151535	-0.151535	0.3073
AvgPerVisit	0.468565	0.629107	-0.206353	1.000000	0.203343	-0.203343	-0.3067
Gender_F	-0.036490	0.161022	0.151535	0.203343	1.000000	-1.000000	-0.2197
Gender_M	0.036490	-0.161022	-0.151535	-0.203343	-1.000000	1.000000	0.2197
Genre_Action	-0.302256	-0.457274	0.307303	-0.306701	-0.219744	0.219744	1.0000
Genre_Comedy	0.101217	0.021663	-0.035128	-0.119992	0.032774	-0.032774	-0.5070
Genre_Drama	0.213388	0.451453	-0.283266	0.434413	0.195067	-0.195067	-0.5374

In [46]:

```
corr_matrix["Rentals"].sort_values(ascending=False)
```

Out[46]:

```
Rentals          1.000000
Genre_Action     0.307303
Gender_F         0.151535
Genre_Comedy    -0.035128
Gender_M        -0.151535
AvgPerVisit     -0.206353
Income          -0.262472
Genre_Drama     -0.283266
Age             -0.547113
Name: Rentals, dtype: float64
```

The new table can be written into a file using to_csv method:

In [43]:

```
vs_norm.to_csv("Video_Store_Numeric.csv", float_format="%1.2f")
```