In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
%matplotlib inline
#import statsmodels.api as sm
#import statsmodels.stats.api as sms
#from statsmodels.stats.proportion import proportion_confint
#import pylab
#import warnings
#warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [96]:
```
data_1 = pd.read_csv('hcmv-263hxkx-1qhtfgz.txt')
```

In [3]:
```
# from original data set, n = 296 (Palindromes), N = 229354 (Base pairs)
n, N = 296, 229354
```

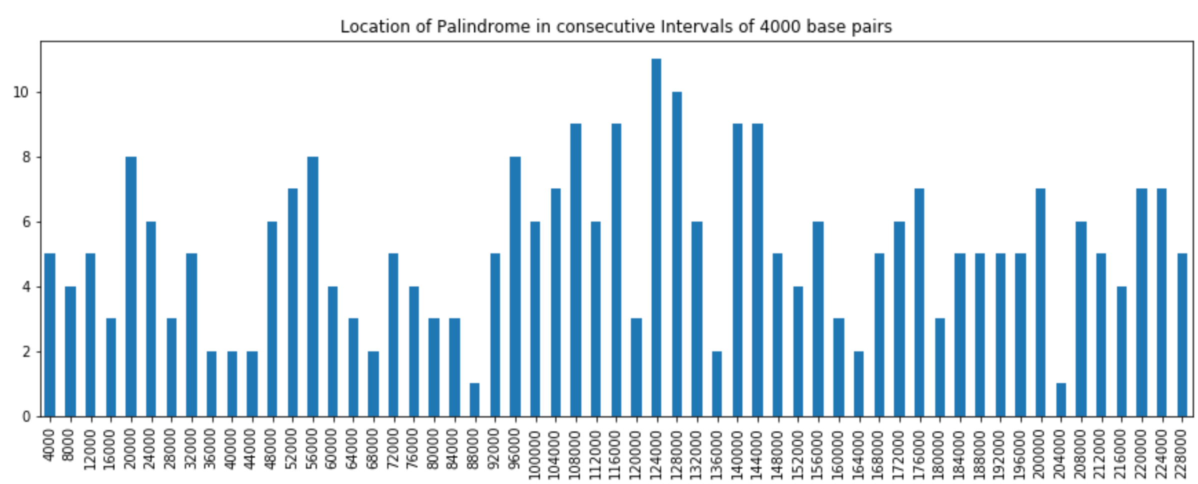
In [4]:
```
# Generate three uniformed distributed
samples = [pd.Series(np.random.uniform(0,N,n)).sort_values() for i in ra
nge(3)]
```

In [ ]:

In [ ]:

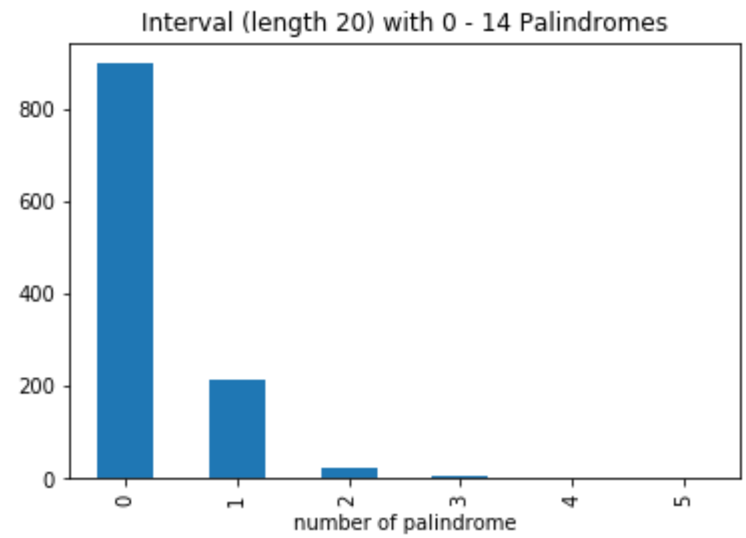
In [5]:
```
pd.cut(samples[0], range(0, N+1, 4000)).apply(lambda x: x.right).value_c
ounts().sort_index()\
.plot(kind = 'bar',figsize=(15,5))
plt.title('Location of Palindrome in consecutive Intervals of 4000 base
 pairs')
```

Out[5]:
```
Text(0.5,1,'Location of Palindrome in consecutive Intervals of 4000 bas
e pairs')
```



In [15]:
```
pd.cut(data_1.location, range(0, N+1, 200)).value_counts().rename('numbe
r of palindrome')\
    .to_frame().groupby('number of palindrome')['number of palindrome'].
count().plot(kind = 'bar')
plt.title('Interval (length 20) with 0 - 14 Palindromes')
```

Out[15]: Text(0.5,1,'Interval (length 20) with 0 - 14 Palindromes')

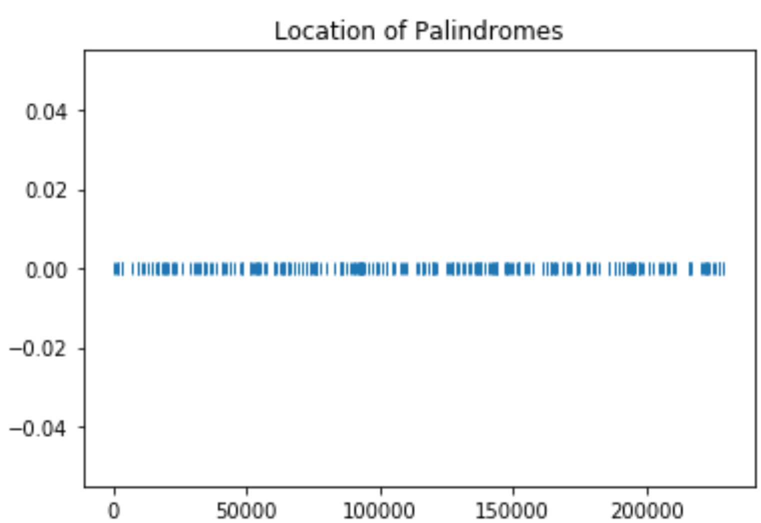


In [ ]:

Random Scatter

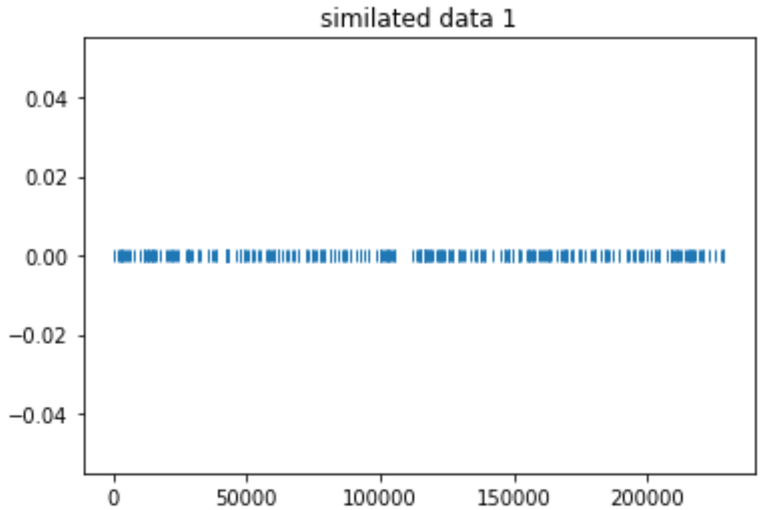
In [5]:
```
#'Location of Palindromes'
plt.plot(data_1.location, np.zeros_like(data_1.location) + 0, '|')
plt.title('Location of Palindromes')
```

Out[5]: Text(0.5,1,'Location of Palindromes')



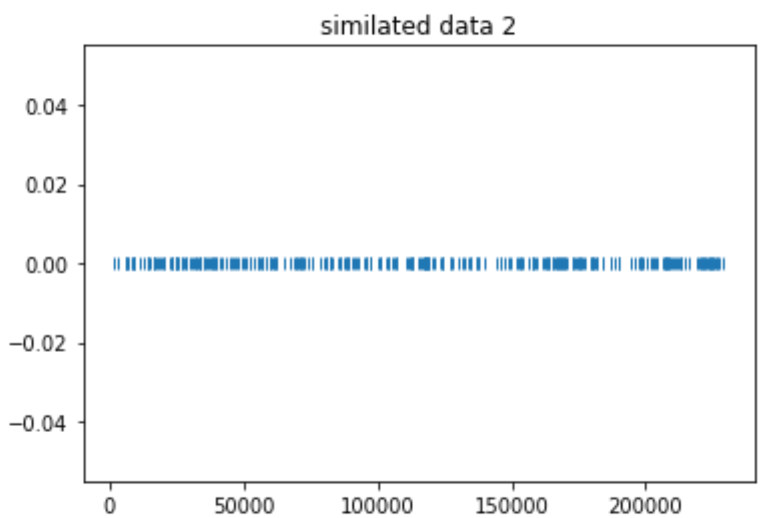
In [6]:
```
# Location of three simulated samples
#for i in samples:
plt.plot(samples[0], np.zeros_like(samples[0]) + 0, '|')
plt.title('similated data 1')
```

Out[6]: Text(0.5,1,'similated data 1')



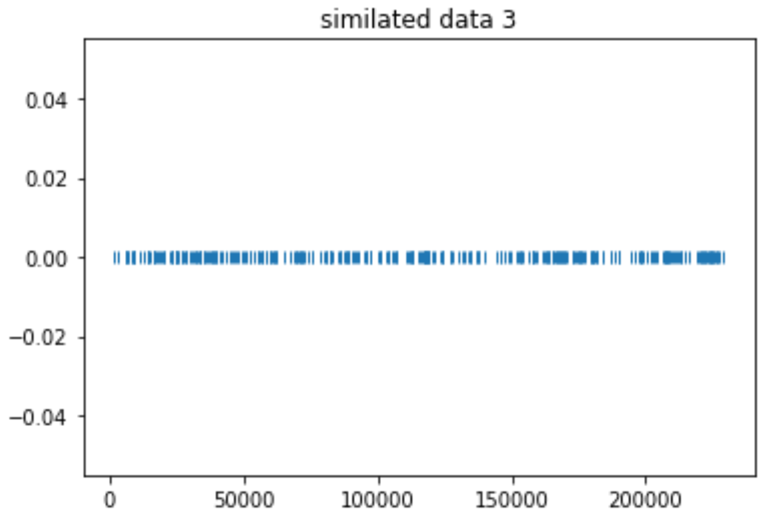
In [7]:
```
plt.plot(samples[1], np.zeros_like(samples[1]) + 0, '|')
plt.title('similated data 2')
```

Out[7]: Text(0.5,1,'similated data 2')



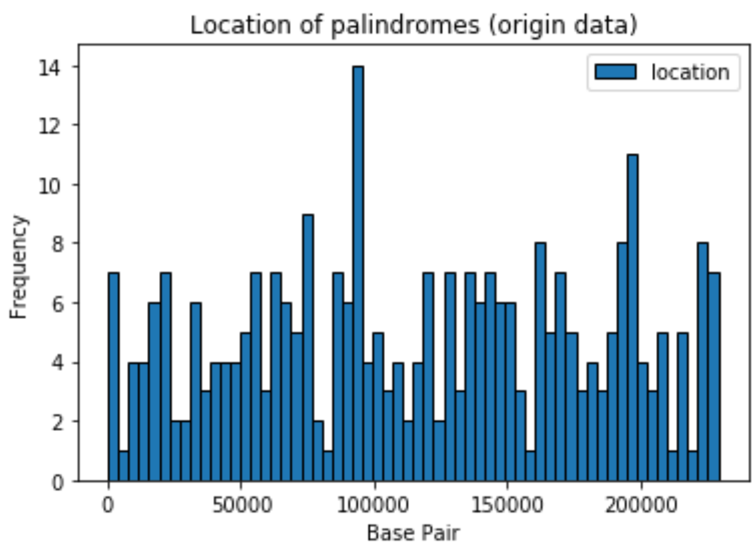
In [8]:
```
plt.plot(samples[1], np.zeros_like(samples[2]) + 0, '|')
plt.title('similated data 3')
```

Out[8]: Text(0.5,1,'similated data 3')



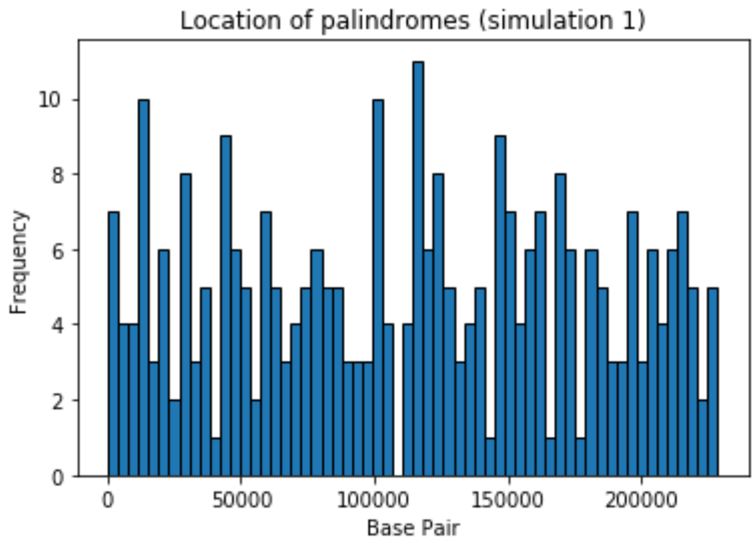
In [9]:
```
data_1.plot(kind="hist", bins = 60, edgecolor = 'k',title='Location of p
alindromes (origin data)')
plt.xlabel("Base Pair")
```

Out[9]: Text(0.5,0,'Base Pair')



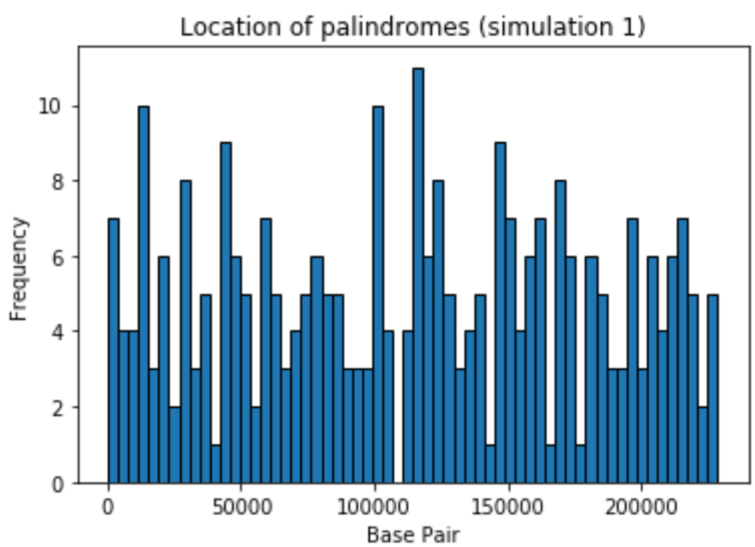
In [10]:
```
pd.Series(samples[0]).sort_values().reset_index()[0].plot(kind= 'hist', bins = 60, edgecolor = 'k',title = ('Location of palindromes (simulation 1)'))
plt.xlabel("Base Pair")
```

Out[10]: Text(0.5,0,'Base Pair')



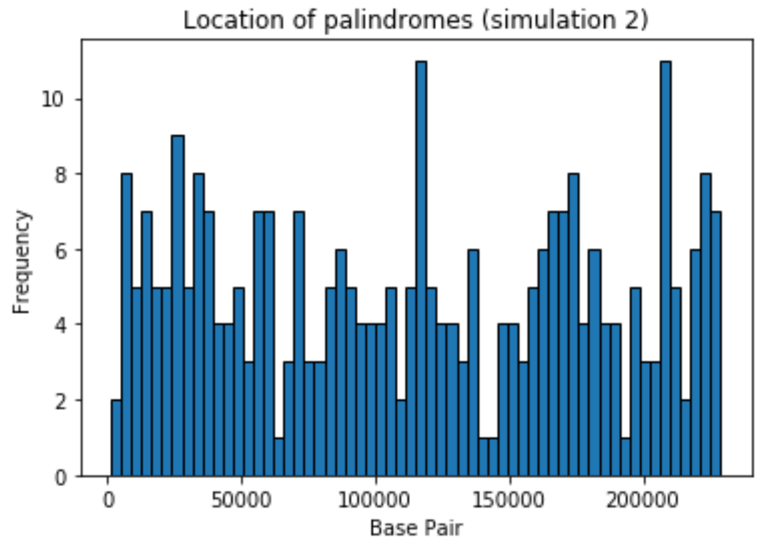
In [11]:
```
pd.Series(samples[0]).plot(kind= 'hist', bins = 60, edgecolor = 'k',title = ('Location of palindromes (simulation 1)'))
plt.xlabel("Base Pair")
```

Out[11]: Text(0.5,0,'Base Pair')



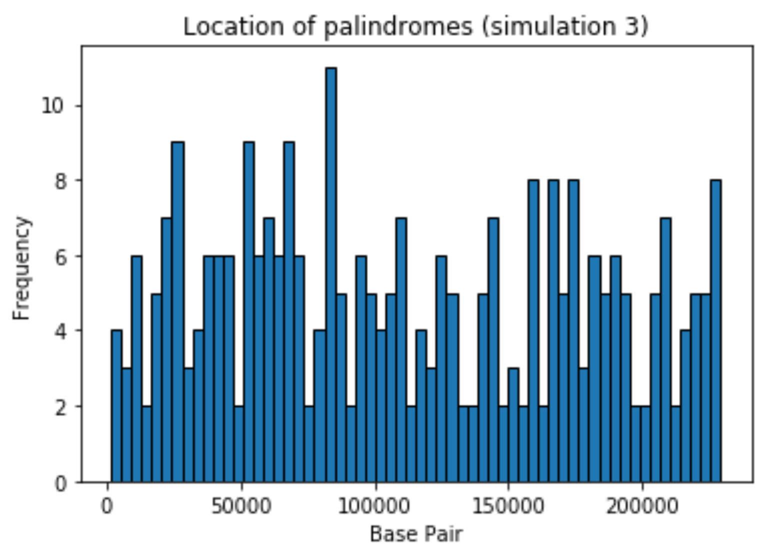
In [12]:
```
pd.Series(samples[1]).plot(kind= 'hist', bins = 60, edgecolor = 'k',titl
e = ('Location of palindromes (simulation 2)'))
plt.xlabel("Base Pair")
```

Out[12]: Text(0.5,0,'Base Pair')



In [13]:
```
pd.Series(samples[2]).plot(kind= 'hist', bins = 60, edgecolor = 'k',titl
e = ('Location of palindromes (simulation 3)'))
plt.xlabel("Base Pair")
```

Out[13]: Text(0.5,0,'Base Pair')

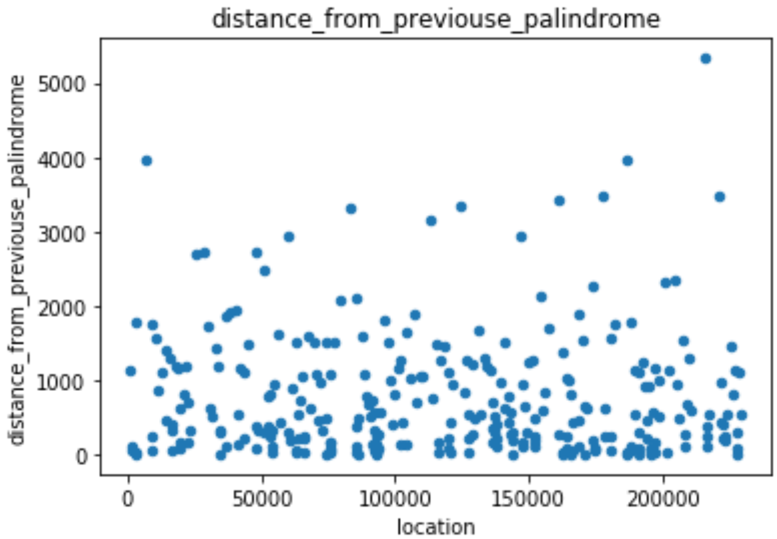


In [14]:
```
data_1 = data_1.assign(distance_from_previouse_palindrome\
                        = data_1.location.diff().set_value(0, np.NaN))
```

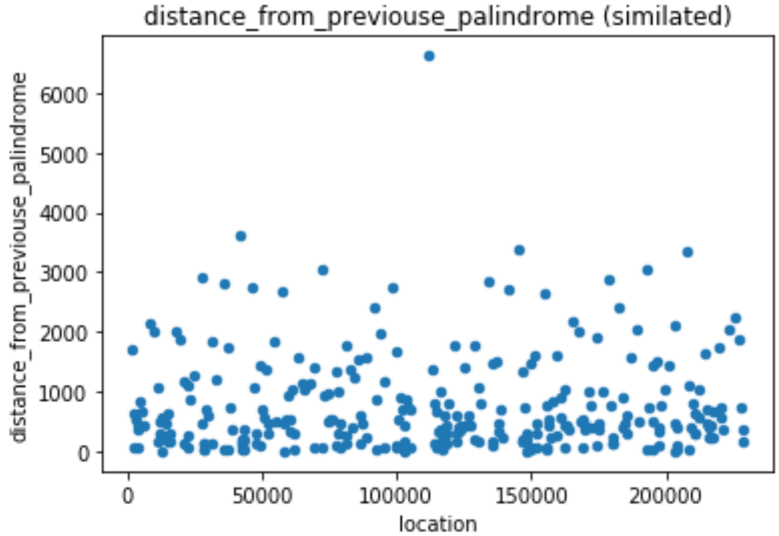
```
/Users/syeehyn/anaconda3/envs/UCSD/lib/python3.6/site-packages/ipykerne
l_launcher.py:2: FutureWarning: set_value is deprecated and will be rem
oved in a future release. Please use .at[] or .iat[] accessors instead
```

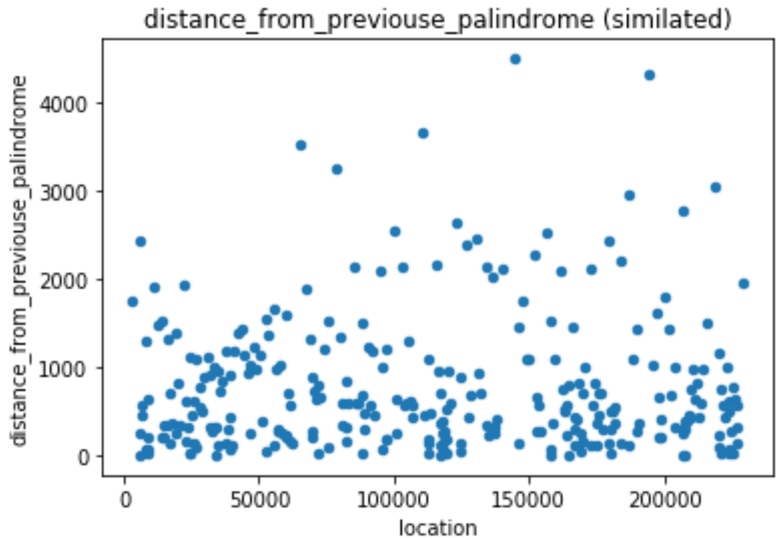
In [15]:
```
data_1.plot(kind = 'scatter', x = 'location', y = 'distance_from_previou
se_palindrome')
plt.title('distance_from_previouse_palindrome')
```

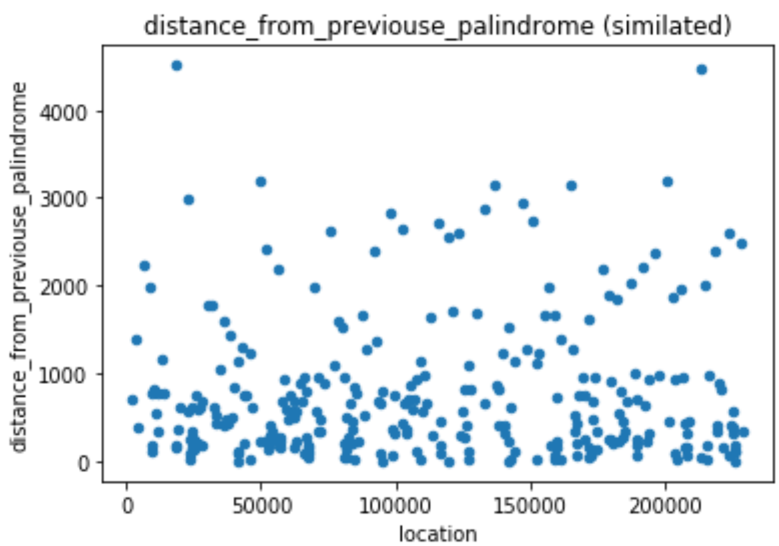
Out[15]:
```
Text(0.5,1,'distance_from_previouse_palindrome')
```



In [16]:
```
df_lst = []
for i in samples:
    df_lst += [
        pd.DataFrame(
                {
                    'location' : i[1:]
                }
        ).assign(distance_from_previouse_palindrome = i.diff().iloc[1:])
\
    ]
for i in df_lst:
    i.plot(kind = 'scatter', x = 'location', y = 'distance_from_previous
e_palindrome')
    plt.title('distance_from_previouse_palindrome (similated)')
```

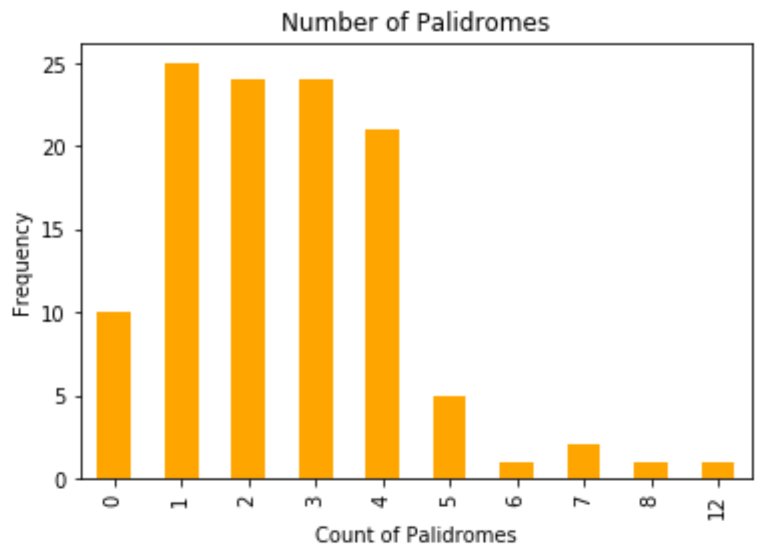

distance_from_previouse_palindrome (similated)
distance_from_previouse_palindrome
location
0
1000
2000
3000
4000
5000
6000
0
50000
100000
150000
200000


distance_from_previouse_palindrome (similated)
distance_from_previouse_palindrome
location
0
1000
2000
3000
4000
0
50000
100000
150000
200000


distance_from_previouse_palindrome (similated)
distance_from_previouse_palindrome
location
0
1000
2000
3000
4000
0
50000
100000
150000
200000

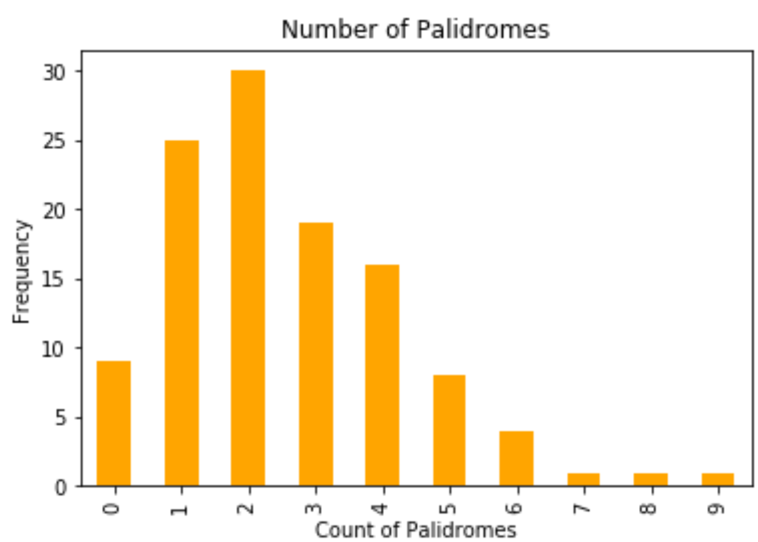
In [17]:
```
original = pd.cut(data_1.location, range(0, N+1, 2000)).value_counts()\
.rename('count').to_frame().groupby('count')['count'].count()
original.plot(kind = 'bar', title="Number of Palidromes", color = 'orang
e')
plt.xlabel("Count of Palidromes")
plt.ylabel("Frequency")
```

Out[17]: Text(0,0.5,'Frequency')



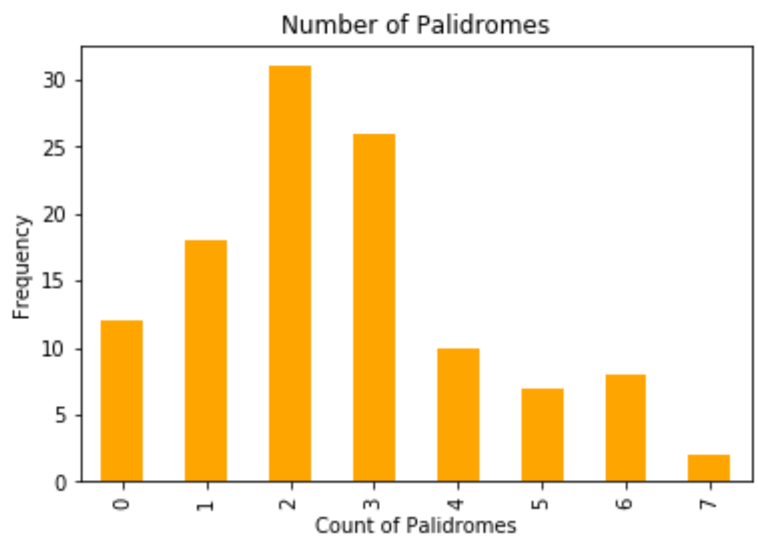
In [18]:
```
pd.cut(samples[0], range(0, N+1, 2000)).value_counts().rename('count')\
    .to_frame().groupby('count')['count'].count().plot(kind = 'bar', tit
le="Number of Palidromes", color = 'orange')
plt.xlabel("Count of Palidromes")
plt.ylabel("Frequency")
```

Out[18]: Text(0,0.5,'Frequency')



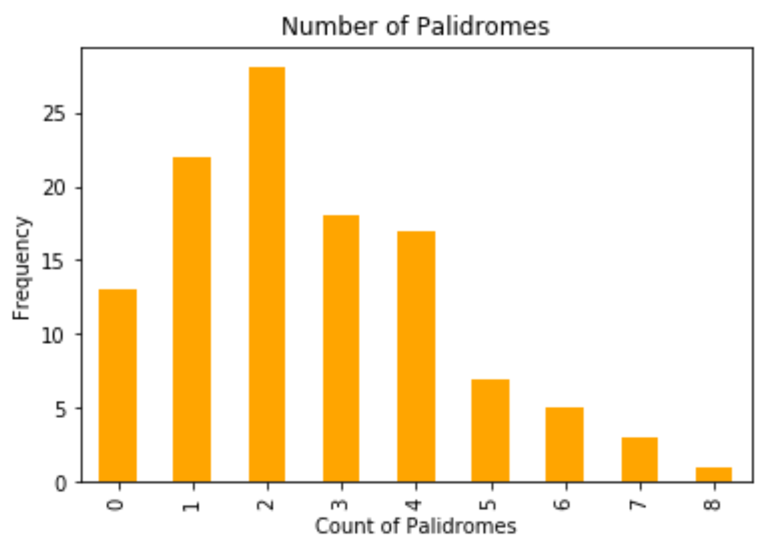
In [19]:
```
pd.cut(samples[1], range(0, N+1, 2000)).value_counts().rename('count')\
    .to_frame().groupby('count')['count'].count().plot(kind = 'bar', tit
le="Number of Palidromes", color = 'orange')
plt.xlabel("Count of Palidromes")
plt.ylabel("Frequency")
```

Out[19]: Text(0,0.5,'Frequency')



In [20]:
```
pd.cut(samples[2], range(0, N+1, 2000)).value_counts().rename('count')\
    .to_frame().groupby('count')['count'].count().plot(kind = 'bar', tit
le="Number of Palidromes", color = 'orange')
plt.xlabel("Count of Palidromes")
plt.ylabel("Frequency")
```

Out[20]: Text(0,0.5,'Frequency')



In [ ]:

Location and Spacing

In [30]:
```
spacing = data_1.location.diff()
spacing_1 = pd.Series(samples[0]).diff()
spacing_2 = pd.Series(samples[1]).diff()
spacing_3 = pd.Series(samples[2]).diff()

spacingdf = pd.DataFrame(
    {
        'original_spacing' : spacing.sort_values().reset_index(drop = Tr
ue),
        'simulated_sample_1' : spacing_1.sort_values().reset_index(drop
= True),
        'simulated_sample_2' : spacing_2.sort_values().reset_index(drop
= True),
        'simulated_sample_3' : spacing_3.sort_values().reset_index(drop
= True),
    }
)
```

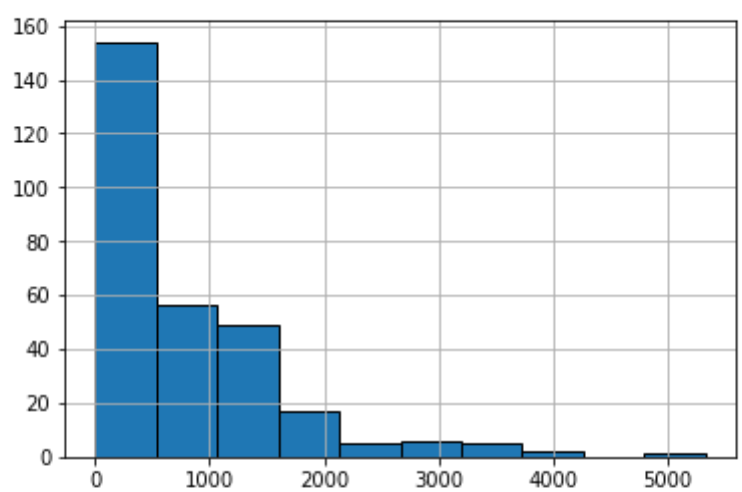
In [31]:
```
spacingdf.head()
```

Out[31]:

	original_spacing	simulated_sample_1	simulated_sample_2	simulated_sample_3
0	1.0	2.069215	0.150657	0.574688
1	5.0	2.515219	0.727582	1.988005
2	5.0	3.468276	0.828988	3.917049
3	6.0	7.693870	0.996015	7.395344
4	6.0	15.863547	3.789403	8.157143

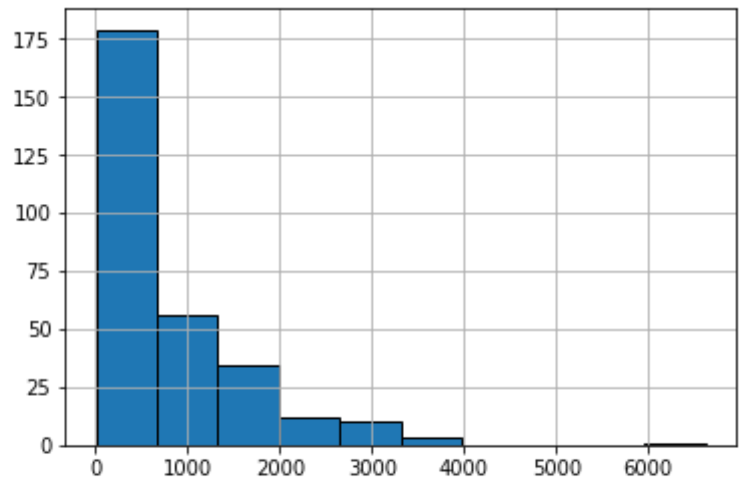
In [32]:
```
spacing.hist(edgecolor = 'k', bins = 10)
#plt.title('spacing histogram of original data set')
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2261fac8>



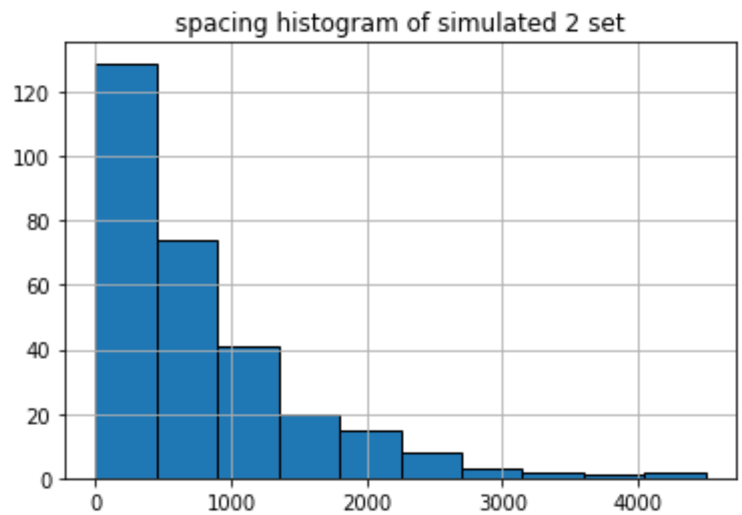
In [33]:
```
spacing_1.hist(edgecolor = 'k', bins = 10)
#plt.title('spacing histogram of simulated 1 set')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a227044a8>



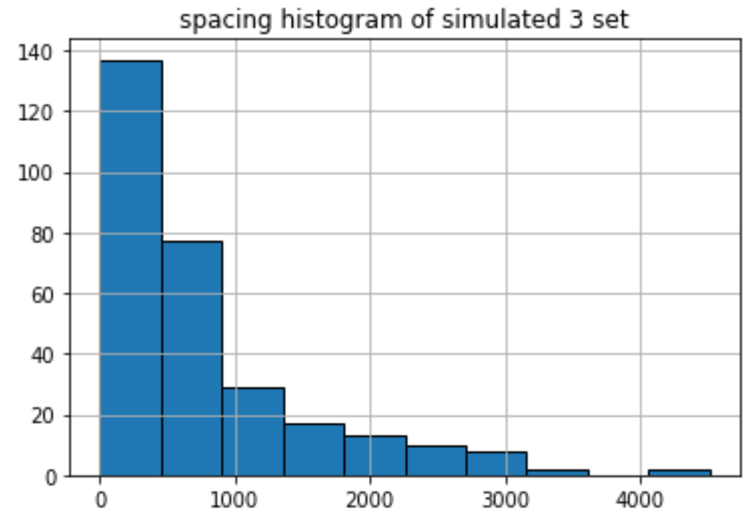
In [34]:
```
spacing_2.hist(edgecolor = 'k')
plt.title('spacing histogram of simulated 2 set')
```

Out[34]: Text(0.5,1,'spacing histogram of simulated 2 set')



In [35]:
```
spacing_3.hist(edgecolor = 'k')
plt.title('spacing histogram of simulated 3 set')
```

Out[35]:
```
Text(0.5,1,'spacing histogram of simulated 3 set')
```



In [36]:
```
#chi-square test
```

In [95]:
```
chisquare_pool = []
for i in range(4):
    chisquare_pool += [spacingdf.iloc[:,i]/10]
    #chisquare_pool += [pd.cut(spacingdf.iloc[:,i], range(0, 5501, 50)).
apply(lambda x: x.right)]
```

In [96]:
```
for i in range(1,4):
    print(stats.chisquare(chisquare_pool[0].dropna(), chisquare_pool[i].
dropna()))
```

```
Power_divergenceResult(statistic=384.24061324216177, pvalue=0.000308635
1443201081)
Power_divergenceResult(statistic=428.1581850767412, pvalue=5.0764502145
65519e-07)
Power_divergenceResult(statistic=481.11042854318333, pvalue=3.268524524
159802e-11)
```

In [41]:
```
# Sum of two consecutive pairs, and spacing.
```

In [42]:
```
diff_pair_table = pd.DataFrame(
    {'spacing of original data distance two pairs':data_1.location.iloc[
range(0,len(data_1),2)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_1 data distance two pairs': samples[0].iloc[rang
e(0,len(samples[0]),2)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_2 data distance two pairs': samples[1].iloc[rang
e(0,len(samples[1]),2)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_3 data distance two pairs': samples[2].iloc[rang
e(0,len(samples[2]),2)].diff().sort_values().reset_index(drop = True)
    })
```

In [43]:
```
diff_pair_table.head()
```

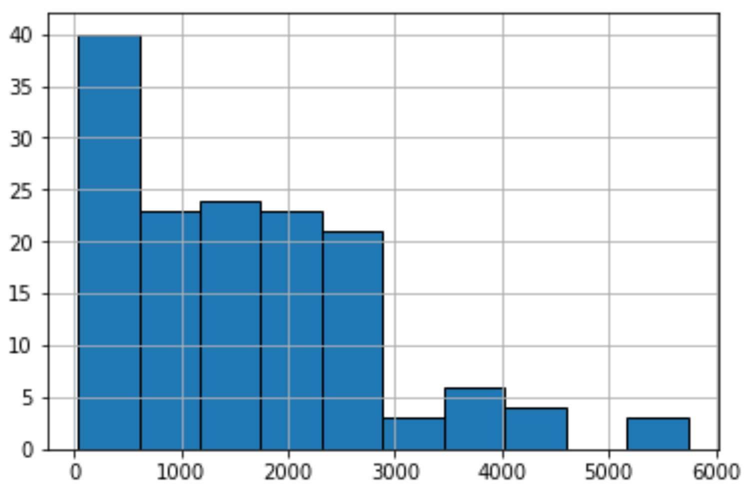
Out[43]:

	spacing of original data distance two pairs	spacing of sample_1 data distance two pairs	spacing of sample_2 data distance two pairs	spacing of sample_3 data distance two pairs
0	38.0	92.432669	26.953491	86.209950
1	39.0	93.286902	71.831406	103.314287
2	46.0	138.480830	87.550081	120.300769
3	48.0	141.311440	209.049002	151.041726
4	77.0	150.954169	252.936388	194.251758

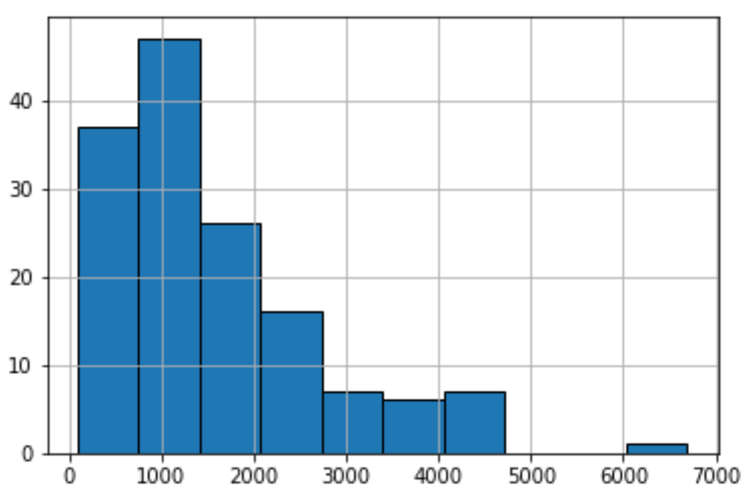
In [44]:
```
diff_pair_table.columns[0]
```

Out[44]: 'spacing of original data distance two pairs'

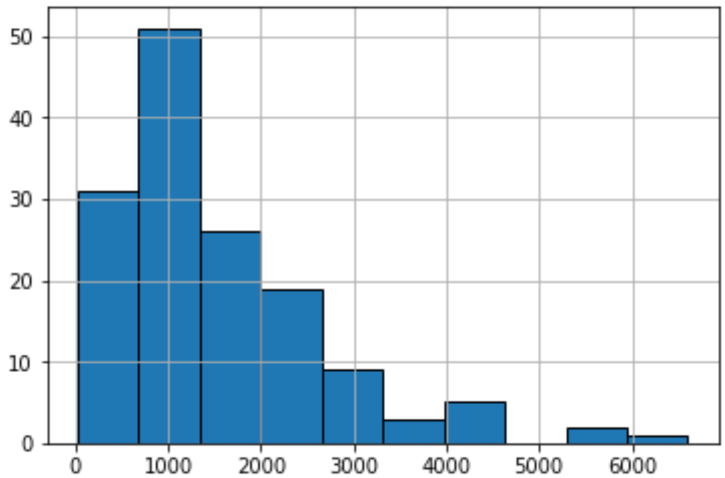
In [45]:
```
diff_pair_table.dropna().iloc[:,0].hist(edgecolor = 'k')
plt.title = (diff_pair_table.columns[0])
```



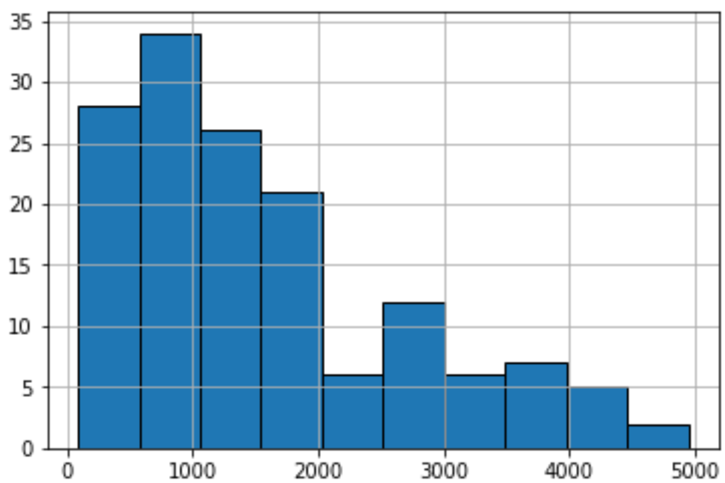
In [46]:
```
diff_pair_table.dropna().iloc[:,1].hist(edgecolor = 'k')
plt.title = (diff_pair_table.columns[1])
```



In [47]:
```
diff_pair_table.dropna().iloc[:,2].hist(edgecolor = 'k')
plt.title = (diff_pair_table.columns[2])
```



In [48]:
```
diff_pair_table.dropna().iloc[:,3].hist(edgecolor = 'k')
plt.title = (diff_pair_table.columns[3])
```



In [49]:
```
chisquare_pool = []
for i in range(4):
    chisquare_pool += [diff_pair_table.iloc[:,i]/10]
    #chisquare_pool += [pd.cut(diff_pair_table.iloc[:,i], range(0, 600,
 500)).apply(lambda x: x.right)]
```

In [50]:
```
#chi-square test pairs
for i in range(1,4):
    print(stats.chisquare(chisquare_pool[0].dropna(), chisquare_pool[i].
dropna()))
```

```
Power_divergenceResult(statistic=490.46073724428703, pvalue=8.129807811
07305e-39)
Power_divergenceResult(statistic=560.6054022716725, pvalue=6.8623605836
98973e-50)
Power_divergenceResult(statistic=401.43936649405805, pvalue=1.045047574
5456958e-25)
```

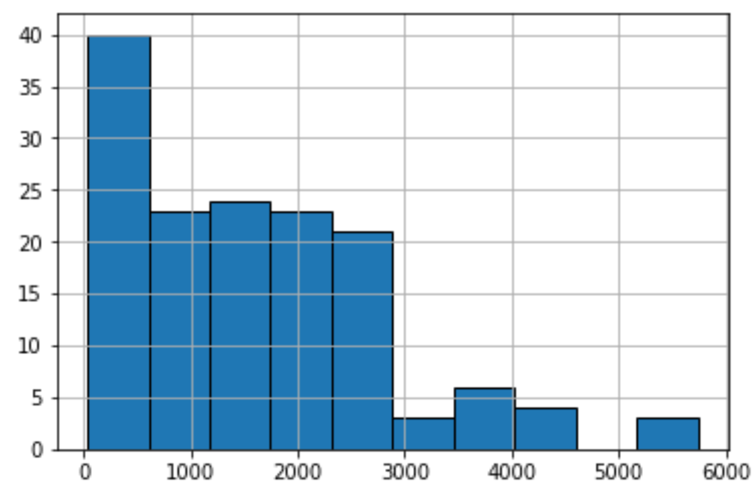
In [51]:
```
diff_pair_table_1 = pd.DataFrame(
    {'spacing of original data distance three pairs':data_1.location.ilo
c[range(0,len(data_1),3)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_1 data distance three pairs': samples[0].iloc[ra
nge(0,len(samples[0]),3)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_2 data distance three pairs': samples[1].iloc[ra
nge(0,len(samples[1]),3)].diff().sort_values().reset_index(drop = True),
     'spacing of sample_3 data distance three pairs': samples[2].iloc[ra
nge(0,len(samples[2]),3)].diff().sort_values().reset_index(drop = True)
    })
```

In [52]:
```
diff_pair_table_1.head()
```

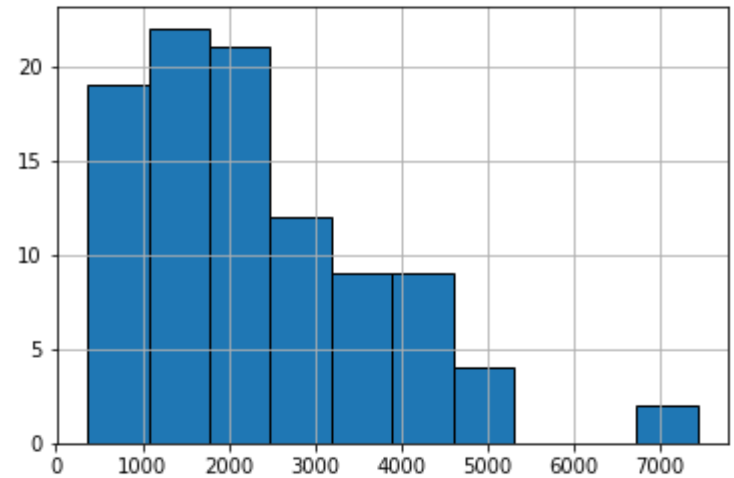
Out[52]:

	spacing of original data distance three pairs	spacing of sample_1 data distance three pairs	spacing of sample_2 data distance three pairs	spacing of sample_3 data distance three pairs
0	109.0	363.577638	289.190892	280.488765
1	139.0	379.409156	499.556538	331.352884
2	150.0	563.576203	638.575908	377.052167
3	256.0	584.022688	678.890549	408.246189
4	380.0	716.939085	679.389147	436.102848

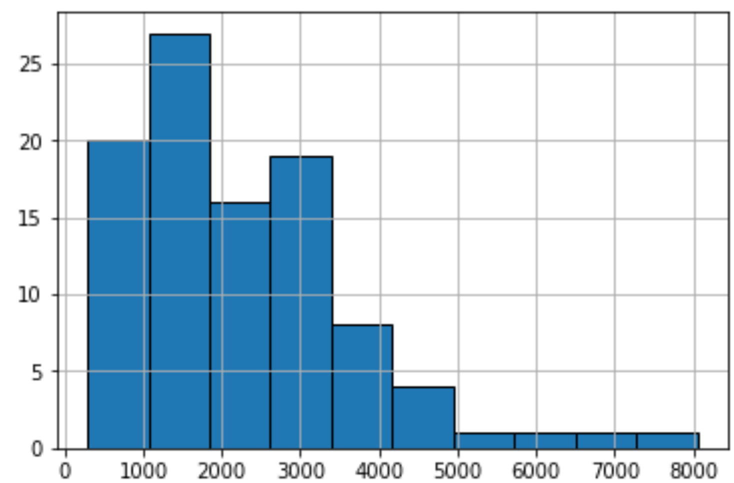
In [53]:
```
diff_pair_table.dropna().iloc[:,0].hist(edgecolor = 'k')
plt.title = (diff_pair_table_1.columns[0])
```



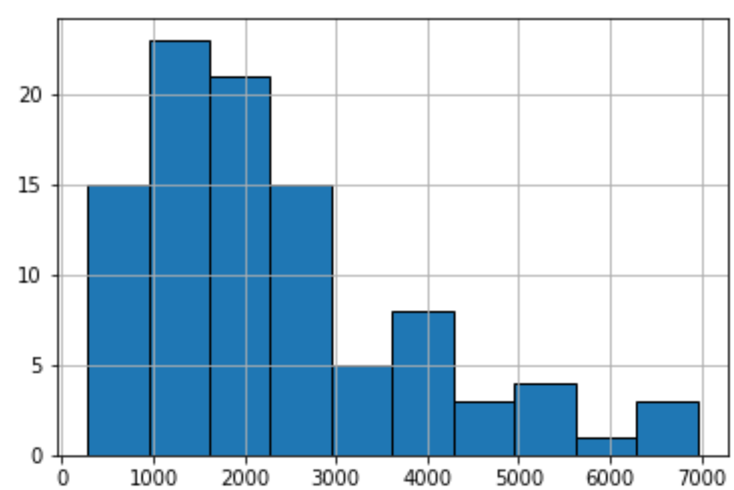
In [54]:
```
diff_pair_table_1.dropna().iloc[:,1].hist(edgecolor = 'k')
plt.title = (diff_pair_table_1.columns[1])
```



In [55]:
```
diff_pair_table_1.dropna().iloc[:,2].hist(edgecolor = 'k')
plt.title = (diff_pair_table_1.columns[2])
```



In [56]:
```
diff_pair_table_1.dropna().iloc[:,3].hist(edgecolor = 'k')
plt.title = (diff_pair_table_1.columns[3])
```



In [57]:
```
chisquare_pool = []
for i in range(4):
    chisquare_pool += [diff_pair_table_1.iloc[:,i]/10]
```

In [58]:
```
#chi-square test for triplet
for i in range(1,4):
    print(stats.chisquare(chisquare_pool[0].dropna(), chisquare_pool[i].
dropna()))
```

```
Power_divergenceResult(statistic=273.19964544021474, pvalue=1.097087429
270108e-18)
Power_divergenceResult(statistic=267.53942255474453, pvalue=6.953761389
774984e-18)
Power_divergenceResult(statistic=149.94293568604348, pvalue=0.000455988
18762795663)
```

In [ ]:

Counts

In [ ]:

In [48]:
```
sample = samples[0] #pick our first random sample as our sample
```

In [49]:
```
for i in range(2000, 5001, 1000):
    pd.cut(data_1.location, range(0, N+1, i)).value_counts().to_frame().
head(10)
```

In [50]:
```
idx = [j for i in list(zip(['orginial_data interval = {}'.format(i) for
i in range(2000, 5001, 1000)],\
    ['simulated_data interval = {}'.format(i) for i in range(2000, 5001,
1000)])) for j in i]
```

In [51]:
```
output = pd.DataFrame(index = idx, columns = range(1, 21))
```

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

In [2]:
```
data_1 = pd.read_csv('hcmv-263hxkx-1qhtfgz.txt')
```

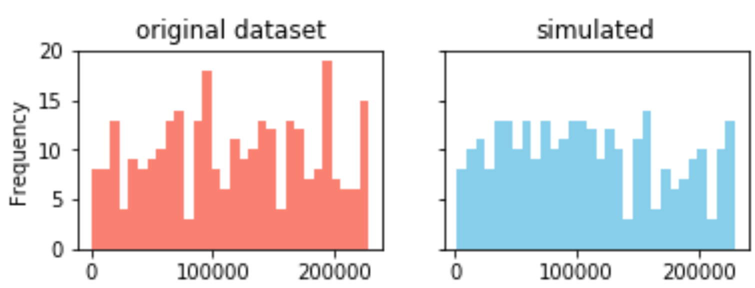
In [3]:
```
# from original data set, n = 296 (Palindromes), N = 229354 (Base pairs)
n, N = 296, 229354
```

In [4]:
```
# Generate three uniformed distributed
samples = [pd.Series(np.random.uniform(0,N,n)).sort_values() for i in ra
nge(3)]
```

Location & Spacing

In [31]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=30, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'salmon')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=30, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(223)
#pd.Series(samples[1]).plot(kind='hist',bins=30, ylim=(0,20), title = 's
imulated 2', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(224)
#pd.Series(samples[2]).plot(kind='hist',bins=30, ylim=(0,20), title = 's
imulated 3', sharex = True, sharey= True, color = 'skyblue')
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a229b4c18>
```

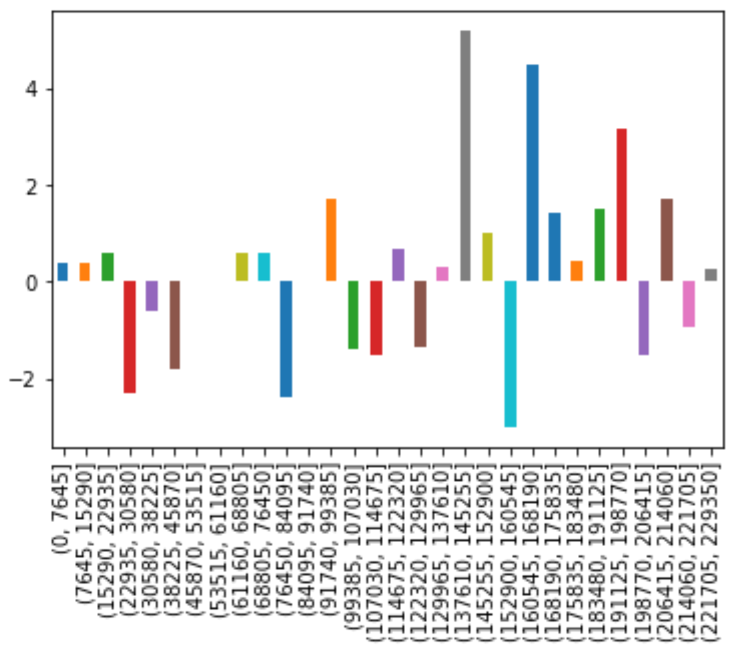


In [68]:
```
orig_i30=pd.cut(data_1.location, bins = range(0, N, N//30)).value_counts
()
sim_i30 = pd.cut(samples[0], bins = range(0, N, N//30)).value_counts()
```

In [69]:
```
res30 = pd.DataFrame({
    'orig': orig_i30,
    'sim': sim_i30
})
```

In [70]:
```
res30.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

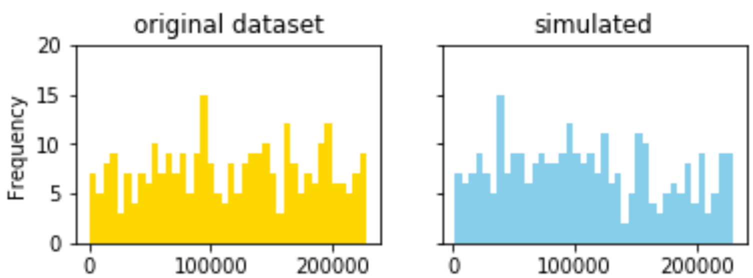
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1107fda90>



In [ ]:

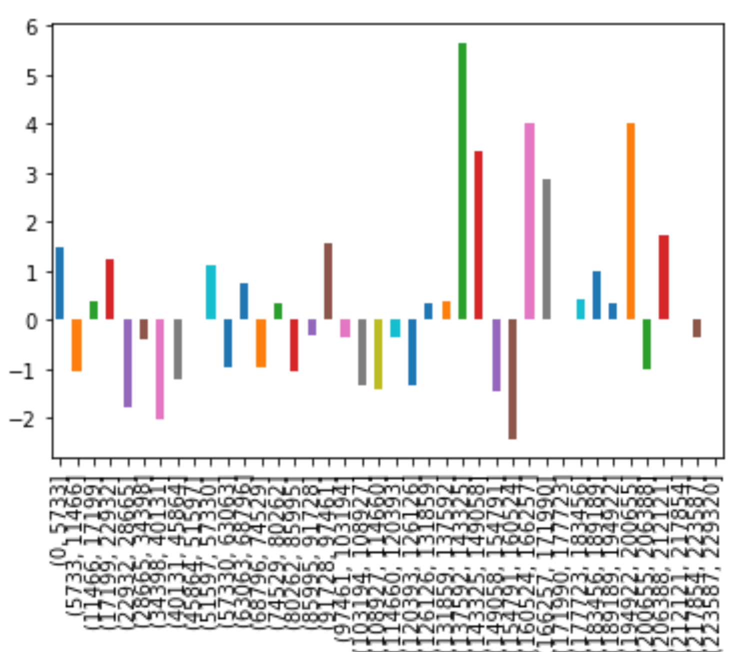
In [32]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=40, ylim=(0,20), title = 'original dataset', sharex = True, sharey= True, color = 'gold')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=40, ylim=(0,20), title = 'simulated', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(223)
#pd.Series(samples[1]).plot(kind='hist',bins=40, ylim=(0,20), title = 'simulated 2', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(224)
#pd.Series(samples[2]).plot(kind='hist',bins=40, ylim=(0,20), title = 'simulated 3', sharex = True, sharey= True, color = 'skyblue')
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1a22a91c18>



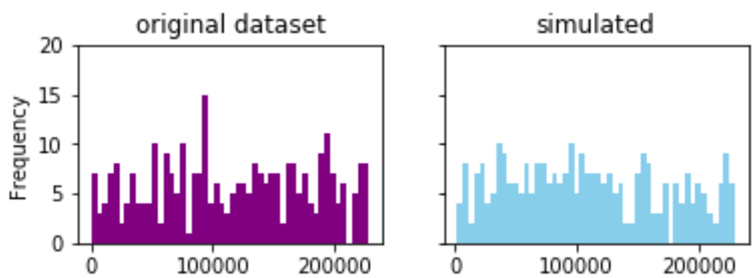
In [67]:
```
orig_i40=pd.cut(data_1.location, bins = range(0, N, N//40)).value_counts()
sim_i40 = pd.cut(samples[0], bins = range(0, N, N//40)).value_counts()
res40 = pd.DataFrame({
    'orig': orig_i40,
    'sim': sim_i40
})
res40.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind = 'bar')
```

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x1a241be940>



In [33]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=50, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'purple')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=50, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(223)
#pd.Series(samples[1]).plot(kind='hist',bins=50, ylim=(0,20), title = 's
imulated 2', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(224)
#pd.Series(samples[2]).plot(kind='hist',bins=50, ylim=(0,20), title = 's
imulated 3', sharex = True, sharey= True, color = 'skyblue')
```

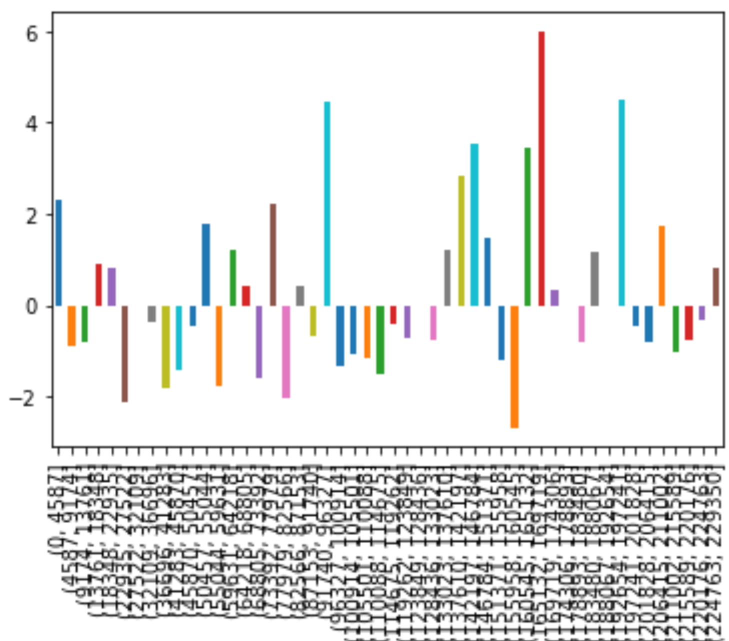
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a22c25940>



In [71]:
```
orig_i50=pd.cut(data_1.location, bins = range(0, N, N//50)).value_counts
()
sim_i50 = pd.cut(samples[0], bins = range(0, N, N//50)).value_counts()
res50 = pd.DataFrame({
    'orig': orig_i50,
    'sim': sim_i50
})
res50.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: Runtime
Warning: divide by zero encountered in true_divide
  import sys
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2444dcf8>



In [74]:
```
from scipy.stats import chisquare
print('with #interval = 30, performing chi-square test, p-val is: ')
print(chisquare(res30.orig, res30.sim))
```

```
with #interval = 30, performing chi-square test, p-val is:
Power_divergenceResult(statistic=103.73113275613277, pvalue=2.452971852
5712304e-10)
```

In [76]:
```
print('with #interval = 40, performing chi-square test, p-val is: ')
print(chisquare(res40.orig, res40.sim))
```

```
with #interval = 40, performing chi-square test, p-val is:
Power_divergenceResult(statistic=125.00010822510822, pvalue=6.109891996
538827e-11)
```

In [78]:
```
print('with #interval = 50, performing chi-square test, p-val is: ')
res50 = res50[res50['sim']!=0]
print(chisquare(res50.orig, res50.sim))
```

```
with #interval = 50, performing chi-square test, p-val is:
Power_divergenceResult(statistic=175.77052669552668, pvalue=1.815456407
8507857e-16)
```

In [ ]:

In [ ]:

In [52]:
```
for i in range(2000, 5001, 1000):
    original = pd.cut(data_1.location, range(0, N+1, i)).value_counts().
rename('count')\
    .to_frame().groupby('count')['count'].count()
    simulated = pd.cut(sample, range(0, N+1, i)).value_counts().rename(
'count')\
    .to_frame().groupby('count')['count'].count()
    for j in original.index:
        output.loc['orginial_data interval = {}'.format(i),j] = original
.loc[j]
    for j in simulated.index:
        output.loc['simulated_data interval = {}'.format(i),j] = simulat
ed.loc[j]
```

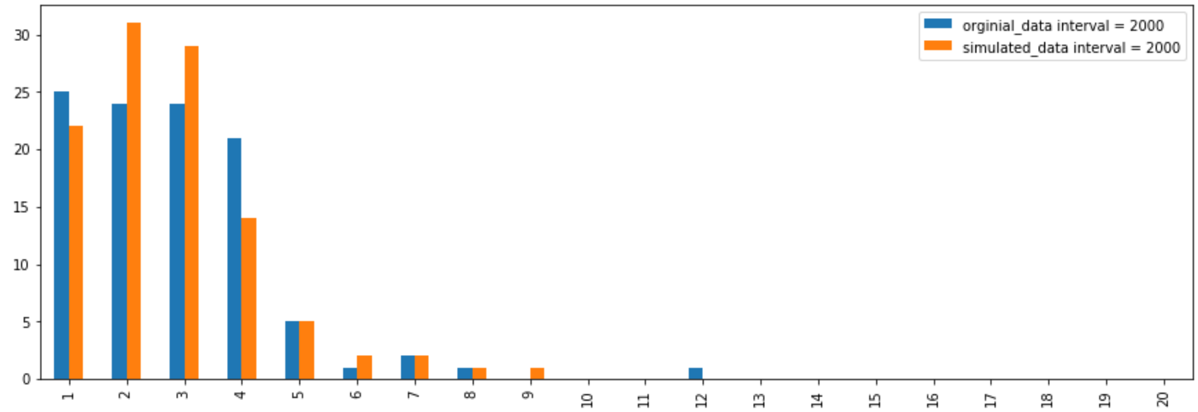
In [53]:
```
output = output.drop(0, axis = 1).fillna(0)
```

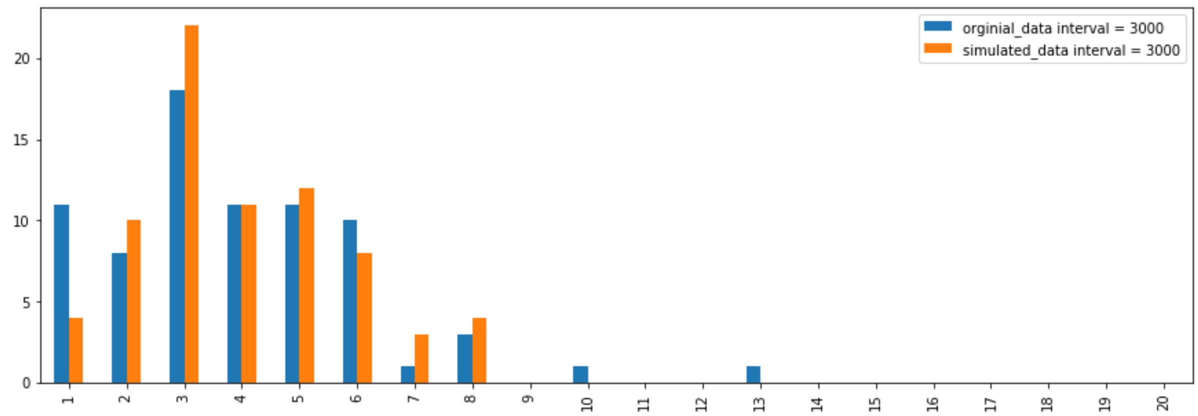
In [54]:
```
output
```

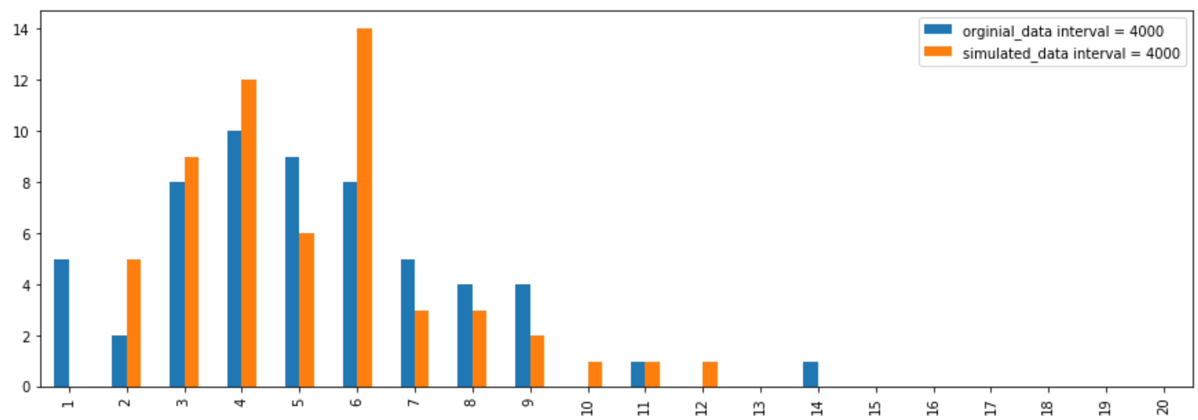
Out[54]:

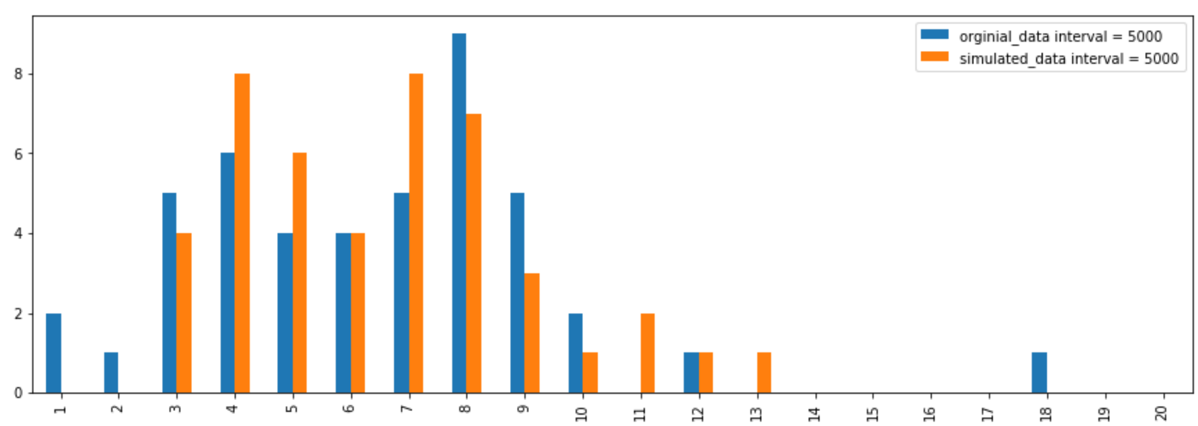
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
orginial_data interval = 2000	25	24	24	21	5	1	2	1	0	0	0	1	0	0	0	0	0	0	0	0
simulated_data interval = 2000	22	31	29	14	5	2	2	1	1	0	0	0	0	0	0	0	0	0	0	0
orginial_data interval = 3000	11	8	18	11	11	10	1	3	0	1	0	0	1	0	0	0	0	0	0	0
simulated_data interval = 3000	4	10	22	11	12	8	3	4	0	0	0	0	0	0	0	0	0	0	0	0
orginial_data interval = 4000	5	2	8	10	9	8	5	4	4	0	1	0	0	1	0	0	0	0	0	0
simulated_data interval = 4000	0	5	9	12	6	14	3	3	2	1	1	1	0	0	0	0	0	0	0	0
orginial_data interval = 5000	2	1	5	6	4	4	5	9	5	2	0	1	0	0	0	0	0	1	0	0
simulated_data interval = 5000	0	0	4	8	6	4	8	7	3	1	2	1	1	0	0	0	0	0	0	0

In [55]:
```
for i in range(0,len(output),2):
#     fig = plt.figure() # Create matplotlib figure
#     ax = fig.add_subplot(111) # Create matplotlib axes
#     ax2 = ax.twinx() # Create another axes that shares the same x-axis
as ax.
#     width = 0.4



    t1 = output.iloc[i:i+2]
    #t2 = output.iloc[i+1]
    t1.T.plot(kind='bar', figsize=(15,5))
    #t2.plot(kind='bar', figsize=(15,5))
    plt.show()
```









In [56]:
```
#Performing chi-square test for count
from scipy.stats import chisquare
```

In [57]:
```
obs = output.iloc[0:11:2]
exp = output.iloc[1:11:2]
```

In [58]:
```
# We choosing the first 0 - 8 counts to roughly enlimulate outliers, mak
ing the chi-suqure test not too greedy
```

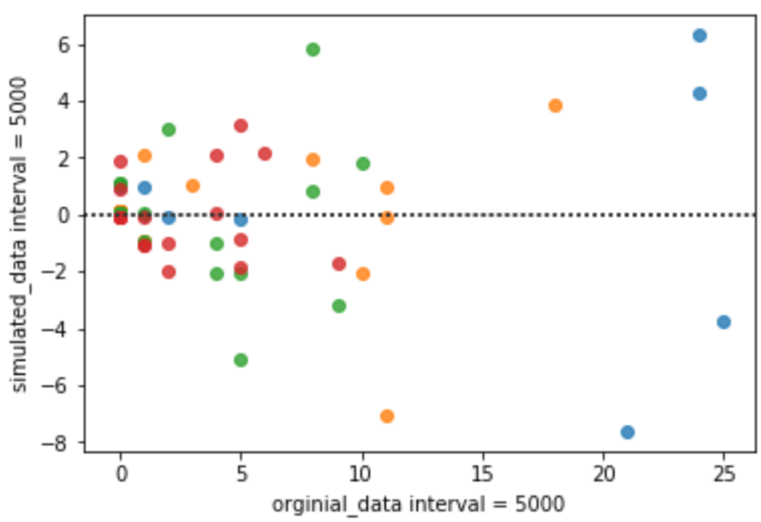
In [62]:
```
counter = [(1,8), (1,8), (1,8), (2, 11)]
for i in range(len(obs)):
    print('with interval = {}, performing chi-square test, p-val is: '.f
ormat((i+1)*1000 + 1000))
    print(chisquare(obs.iloc[i,counter[i][0]:counter[i][1]], exp.iloc[i,
counter[i][0]:counter[i][1]])[1])
```

```
with interval = 2000, performing chi-square test, p-val is:
0.3754643358182173
with interval = 3000, performing chi-square test, p-val is:
0.7711521463799177
with interval = 4000, performing chi-square test, p-val is:
0.2393852869355991
with interval = 5000, performing chi-square test, p-val is:
0.4893186745810869
```

In [64]:
```
# P-Val is large
```

In [65]:
```
# ploting standardlize residuals
```

In [66]:
```
import seaborn as sns
for i in range(len(obs)):
    sns.residplot(obs.iloc[i], exp.iloc[i])
```



In [ ]:

The Biggest Cluster

In [27]:
```
#bins
bins = [40, 60, 80, 100, 120]
```

In [28]:
```
out = pd.DataFrame(index = bins, columns = ['lambda', 'interval_width',
'probability', 'maximum', 'prediction_interval'])
for i in bins:
    interval = N//i
    k = pd.cut(data_1.location, range(0, N+1, interval)).value_counts().
rename('count').max()
    x_bar = pd.cut(data_1.location, range(0, N+1, interval)).value_count
s().rename('count').mean()
    pred = pd.cut(data_1.location, range(0, N+1, interval)).value_counts
().rename('count').idxmax()
    P = (x_bar**(k)/ np.math.factorial((k))) * np.exp(-x_bar)
    out.loc[i] = x_bar, interval, P, k, pred
```

In [29]:
```
out
```

Out[29]:

	lambda	interval_width	probability	maximum	prediction_interval
40	7.4	5733	0.00510737	15	(91728, 97461]
60	4.93333	3822	0.000417868	14	(91728, 95550]
80	3.7	2866	2.5558e-05	14	(91712, 94578]
100	2.96	2293	1.1143e-05	13	(91720, 94013]
120	2.46667	1911	1.70567e-06	13	(91728, 93639]

In [ ]:

Additional hypothesis

In [93]:
```
# if biologist found this cluster is not significant, we want to exclude
the attribute of this interval,
# and we will analysis the statiscally significance of the second larges
t interval of palindrome
```

In [94]:
```
# We fix our bin with 120, and the prediction_interval to exclude will b
e (91728, 93639]
# We will use possion process to impute the data set a random set
```

In [97]:
```
interval = 120
```

In [98]:
```
additional_data = data_1.copy()
```

In [99]:
```
additional_data.head()
```

Out[99]:

	location
0	177
1	1321
2	1433
3	1477
4	3248

In [100]:
```
additional_data = additional_data.loc[(additional_data.location <= 91728
) | (additional_data.location > 93639)]
```

In [101]:
```
additional_data.shape
```

Out[101]: (283, 1)

In [102]:
```
# Test for location and spacing
```

In [103]:
```
n, N = 283, 229354
```

In [104]:
```
samples = [pd.Series(np.random.uniform(0,N,n)).sort_values() for i in ra
nge(3)]
```

In [105]:
```
data_1 = additional_data
```

In [106]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=30, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'salmon')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=30, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
```

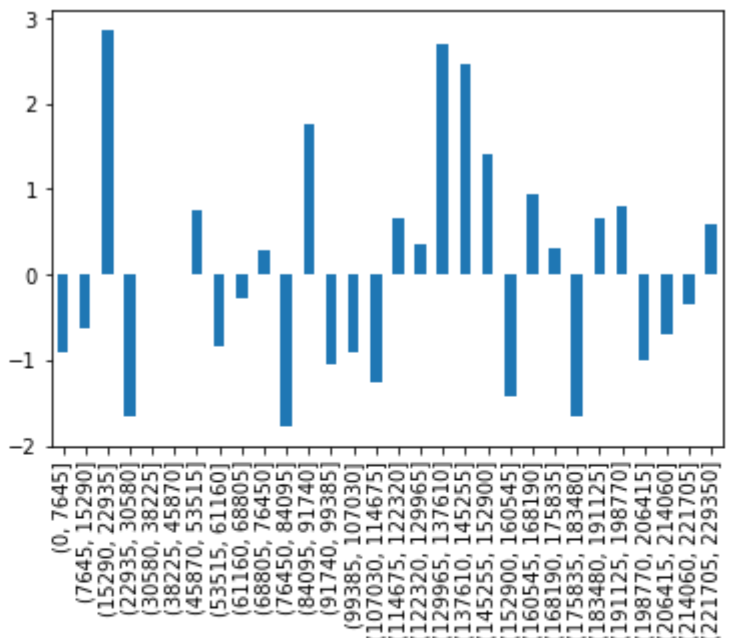
Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19c00e80>

In [107]:
```
orig_i30=pd.cut(data_1.location, bins = range(0, N, N//30)).value_counts
()
sim_i30 = pd.cut(samples[0], bins = range(0, N, N//30)).value_counts()
```

In [108]:
```
res30 = pd.DataFrame({
    'orig': orig_i30,
    'sim': sim_i30
})
```

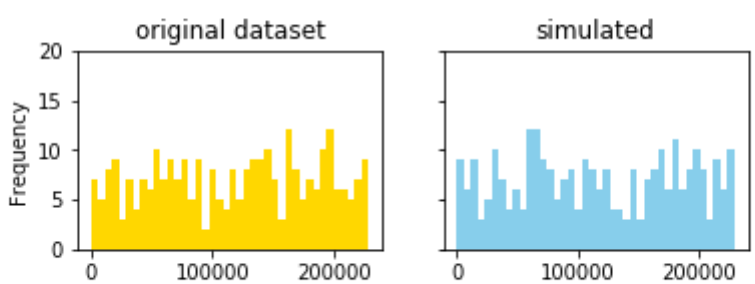
In [109]:
```
res30.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19d5d4a8>



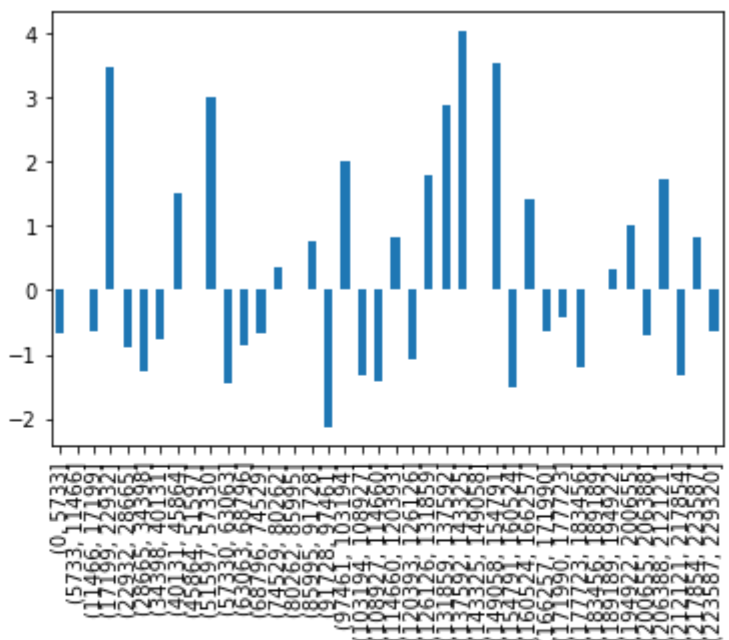
In [110]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=40, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'gold')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=40, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
```

Out[110]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19ebb780>



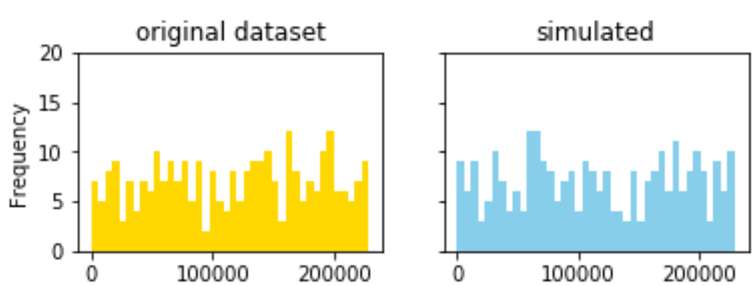
In [111]:
```
orig_i40=pd.cut(data_1.location, bins = range(0, N, N//40)).value_counts
()
sim_i40 = pd.cut(samples[0], bins = range(0, N, N//40)).value_counts()
res40 = pd.DataFrame({
    'orig': orig_i40,
    'sim': sim_i40
})
res40.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

Out[111]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a0555c0>



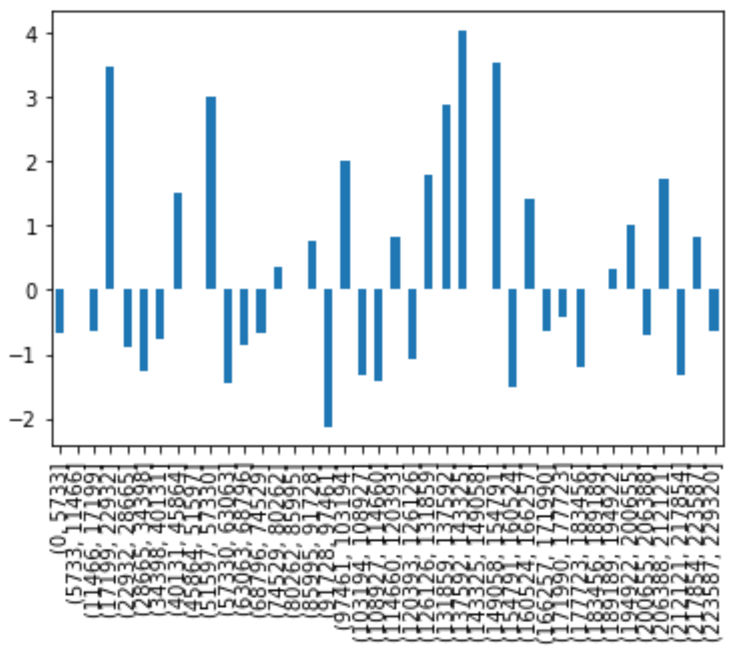
In [112]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=40, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'gold')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=40, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
#plt.subplot(223)
```

Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a2416a0>



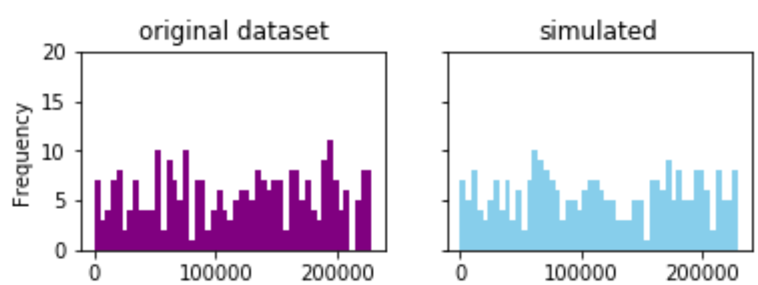
In [113]:
```
orig_i40=pd.cut(data_1.location, bins = range(0, N, N//40)).value_counts
()
sim_i40 = pd.cut(samples[0], bins = range(0, N, N//40)).value_counts()
res40 = pd.DataFrame({
    'orig': orig_i40,
    'sim': sim_i40
})
res40.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a33fdd8>



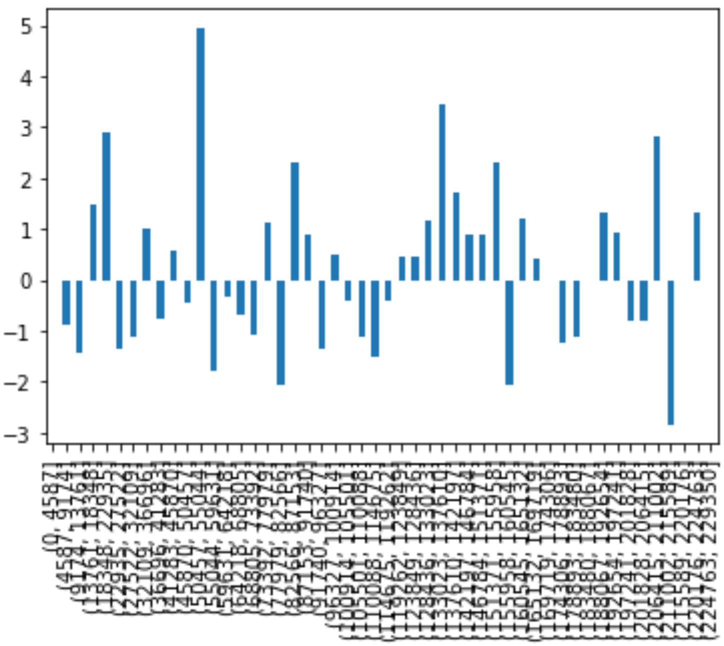
In [114]:
```
plt.subplot(221)
data_1['location'].plot(kind='hist',bins=50, ylim=(0,20), title = 'origi
nal dataset', sharex = True, sharey= True, color = 'purple')
plt.subplot(222)
pd.Series(samples[0]).plot(kind='hist',bins=50, ylim=(0,20), title = 'si
mulated', sharex = True, sharey= True, color = 'skyblue')
```

Out[114]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a4e6d30>



In [115]:
```
orig_i50=pd.cut(data_1.location, bins = range(0, N, N//50)).value_counts
()
sim_i50 = pd.cut(samples[0], bins = range(0, N, N//50)).value_counts()
res50 = pd.DataFrame({
    'orig': orig_i50,
    'sim': sim_i50
})
res50.apply(lambda x: (x.orig - x.sim)/x.sim**(0.5), axis = 1).plot(kind
= 'bar')
```

Out[115]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a6b4b38>



In [116]:
```
from scipy.stats import chisquare
print('with #interval = 30, performing chi-square test, p-val is: ')
print(chisquare(res30.orig, res30.sim))
```

```
with #interval = 30, performing chi-square test, p-val is:
Power_divergenceResult(statistic=47.943115218115224, pvalue=0.014898755
726929839)
```

In [117]:
```
print('with #interval = 40, performing chi-square test, p-val is: ')
print(chisquare(res40.orig, res40.sim))
```

```
with #interval = 40, performing chi-square test, p-val is:
Power_divergenceResult(statistic=99.23867243867244, pvalue=3.7358384318
85617e-07)
```

In [118]:
```
print('with #interval = 50, performing chi-square test, p-val is: ')
res50 = res50[res50['sim']!=0]
print(chisquare(res50.orig, res50.sim))
```

```
with #interval = 50, performing chi-square test, p-val is:
Power_divergenceResult(statistic=119.52777777777779, pvalue=7.956403480
48162e-08)
```

In [119]:
```
bins = [40, 60, 80, 100, 120]
```

In [120]:
```
out = pd.DataFrame(index = bins, columns = ['lambda', 'interval_width',
'probability', 'maximum', 'prediction_interval'])
for i in bins:
    interval = N//i
    k = pd.cut(data_1.location, range(0, N+1, interval)).value_counts().
rename('count').max()
    x_bar = pd.cut(data_1.location, range(0, N+1, interval)).value_count
s().rename('count').mean()
    pred = pd.cut(data_1.location, range(0, N+1, interval)).value_counts
().rename('count').idxmax()
    P = (x_bar**(k)/ np.math.factorial((k))) * np.exp(-x_bar)
    out.loc[i] = x_bar, interval, P, k, pred
```

In [121]:
```
out
```

Out[121]:

	lambda	interval_width	probability	maximum	prediction_interval
40	7.075	5733	0.0277809	12	(194922, 200655]
60	4.71667	3822	0.00226393	12	(194922, 198744]
80	3.5375	2866	0.00245971	10	(194888, 197754]
100	2.83	2293	0.000535856	10	(194905, 197198]
120	2.35833	1911	0.00761368	7	(74529, 76440]

In [ ]: