In [115]:

```python
#CS156 - 10.1.py
%matplotlib inline

"""
See data at: https://gist.github.com/oba2311/c1c63e1f635910e87a4302fdef902f3f
"""

import csv
import numpy as np
import pandas as pd
from sklearn.neighbors.kde import KernelDensity
import matplotlib.pyplot as plt
import random

random.seed(23)

data=pd.read_csv('data.csv',delim_whitespace=True)
data.head()
```

Out[115]:

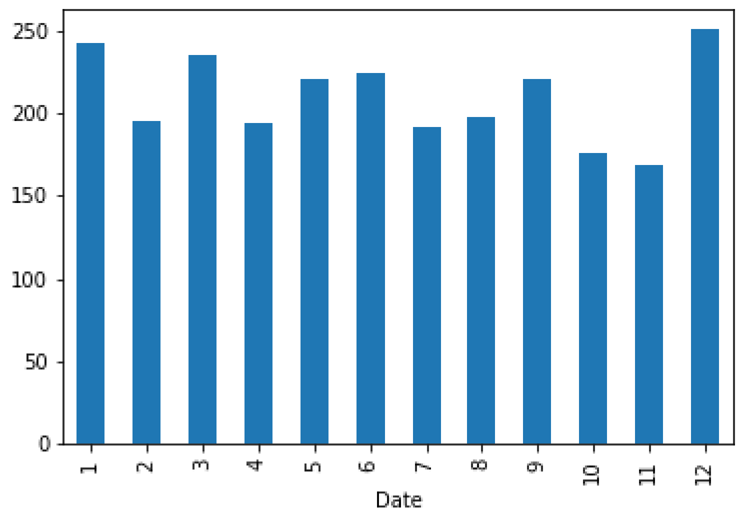
	Date	Amount
0	25.5.2016	54,241.35
1	29.5.2017	54,008.83
2	30.6.2017	54,008.82
3	05.1.2017	52,704.37
4	23.2.2017	52,704.36

In [3]:

```
data['Date']=pd.to_datetime(data['Date'],infer_datetime_format=True)
months = data.groupby(data["Date"].dt.month).count()
plot_object_of_months = data.groupby(data["Date"].dt.month).count()
plot_object_of_months= plot_object_of_months.loc[:,"Amount"]
plot_object_of_months.plot(kind="bar")
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10d77bf90>
```

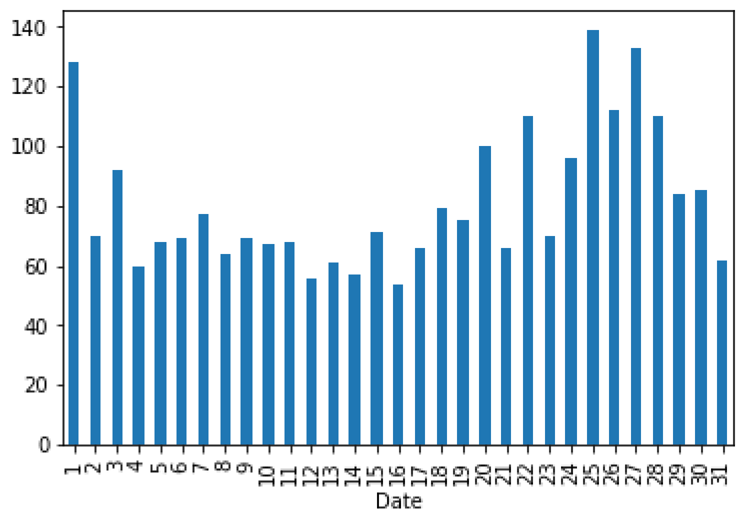


In [125]:

```
data['Date']=pd.to_datetime(data['Date'],infer_datetime_format=True)
plot_object_of_days = data.groupby(data["Date"].dt.day).count()
days = data.groupby(data["Date"].dt.day).count()
plot_object_of_days= plot_object_of_days.loc[:,"Amount"]

plot_object_of_days.plot(kind="bar")
```

Out[125]:

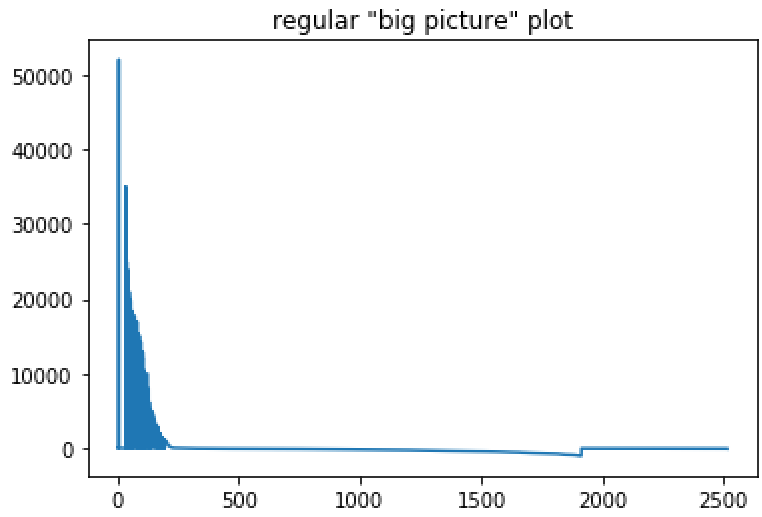
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18d3cf50>
```

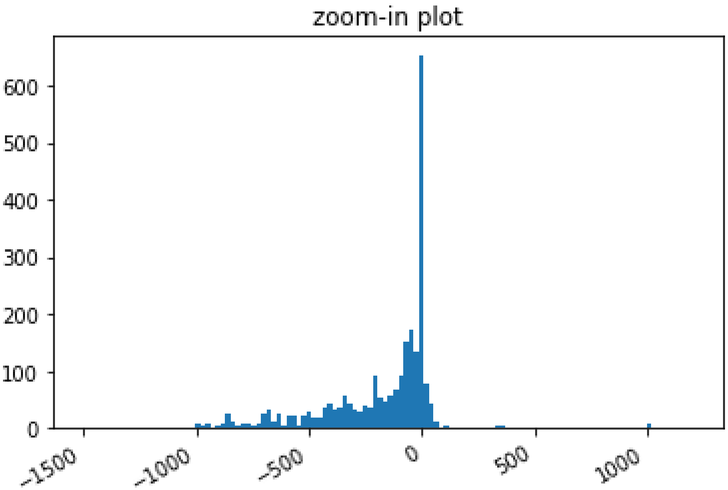


In [126]:

```
amount = data['Amount']
amount = amount.values.reshape(-1,1)

amount_as_num = []
for i in amount:
    i = i[0].replace(',', '.')
    i = np.fromstring(i, dtype=np.float, sep=',' )
    amount_as_num.append(i)

data["amount_as_num"] = amount_as_num

amount_as_num = np.array(amount_as_num).reshape(-1,1)

print data["amount_as_num"].head()

#Here we zoom in to see most of the data. This can be justified by looking at th
plt.plot(amount_as_num)
plt.title("regular \"big picture\" plot")

#Just a regular plotting of the data shows that the data is somewhat weird with

fig, ax = plt.subplots()
#Therefore, an option is to zoom in and try to better understand the area under
# we are taking the risk of generating data that will not fake accuratly outlie
plt.hist(amount_as_num,bins = 120,range=(-1500,1200),align='mid')
plt.title("zoom-in plot")

fig.autofmt_xdate() # make space for and rotate the x-axis tick labels
plt.show()
```

```
0    [54.241]
1    [54.008]
2    [54.008]
3    [52.704]
4    [52.704]
Name: amount_as_num, dtype: object
```





In [127]:

```
kde_month = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(months)
kde_month.score_samples(months)
```

Out[127]:

```
array([-1.10390789, -1.10390789, -1.10390789, -1.10390789, -0.4107607
1,
       -1.10390789, -1.10390789, -1.10390789, -0.41076071, -1.1039078
9,
       -1.10390789, -1.10390789])
```

In [128]:

```
kde_month.sample(10)
```

Out[128]:

```
array([[ 176.37641122,  175.8224081 ],
       [ 190.95049999,  191.2712638 ],
       [ 223.91574546,  223.70898105],
       [ 221.04182044,  220.79627387],
       [ 250.99192953,  250.98516124],
       [ 175.95329179,  175.96377093],
       [ 243.03881323,  242.9664534 ],
       [ 250.91757258,  250.89227902],
       [ 175.67315668,  176.20988767],
       [ 224.07053017,  223.8826399 ]])
```

In [129]:

```
kde_day = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(days)
kde_day.score_samples(days)
```

Out[129]:

```
array([-2.05298845, -1.35984127, -2.05298845, -2.05298845, -1.3598412
7,
       -1.35984127, -2.05298845, -2.05298845, -1.35984127, -2.0529884
5,
       -1.35984127, -2.05298845, -2.05298845, -2.05298845, -2.0529884
5,
       -2.05298845, -1.35984127, -2.05298845, -2.05298845, -2.0529884
5,
       -1.35984127, -1.35984127, -1.35984127, -2.05298845, -2.0529884
5,
       -2.05298845, -2.05298845, -1.35984127, -2.05298845, -2.0529884
5,
       -2.05298845])
```

In [130]:

```
kde_day.sample(10)
```

Out[130]:

```
array([[  67.77015889,   67.54265004],
       [  65.988236  ,   65.70493764],
       [  75.06384941,   75.2135438 ],
       [ 110.02243013,  109.90202193],
       [  56.65548486,   57.17725381],
       [  68.54562583,   69.55292289],
       [  55.68722254,   55.81888602],
       [  60.01100802,   60.0279363 ],
       [  92.16851497,   92.2393542 ],
       [ 128.04089359,  127.93803894]])
```

In [131]:

```
kde_amount = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(amount_as_num)
kde_amount.score_samples(amount_as_num)
kde_amount.sample(10)
```

Out[131]:

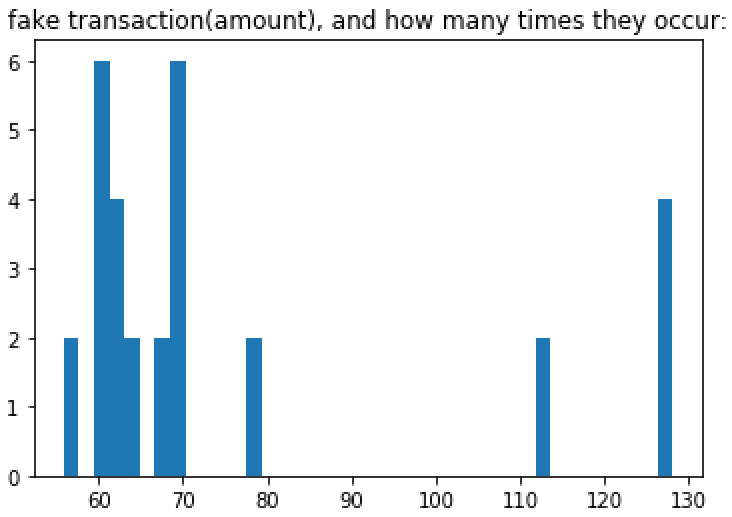
```
array([[ -58.77152887],
       [ -47.14064439],
       [-260.07169609],
       [-164.8674097 ],
       [  -7.85387687],
       [  38.01915293],
       [  -1.1661075 ],
       [  -5.73407325],
       [ -27.20152874],
       [-846.83686649]])
```

Sampling from these density models, create a fictitious month of personal transactions:

In [132]:

```
Month = kde_day.sample(15)
Month=Month.reshape(-1,1)

plt.hist(Month, bins = 40)
plt.title("fake transaction(amount), and how many times they occur:")
print "here's the data:", Month
```

```
here's the data: [[  60.90855642]
 [  61.26446341]
 [ 127.97035796]
 [ 127.85768522]
 [  60.10264321]
 [  60.01020326]
 [  79.24471743]
 [  78.74509348]
 [  66.98870391]
 [  67.08969987]
 [ 111.84698394]
 [ 112.20338867]
 [ 127.96245835]
 [ 128.071591  ]
 [  62.16235396]
 [  62.0289875 ]
 [  55.85694027]
 [  55.86262854]
 [  62.01164561]
 [  61.60400723]
 [  64.35600784]
 [  64.36502073]
 [  70.0263983 ]
 [  70.06341142]
 [  69.87134987]
 [  70.1280842 ]
 [  59.8175005 ]
 [  60.23727729]
 [  68.69229151]
 [  69.2972583 ]]
```



In [149]:

```

print "total number of transactions:" ,np.mean(kde_month.sample(10))
print """
notice that by averaging high numbers of generated transactions, we are getting
central limit theorm. That said, by doing so we are still interpulating in a se
"""
```

```
total number of transactions: 213.734830778

notice that by averaging high numbers of generated transactions, we ar
e getting at a more likely expected number, based on the
central limit theorm. That said, by doing so we are still interpulatin
g in a sense, since the data is still based only on the data we traine
d on. So, the risk of overfitting is real.
```

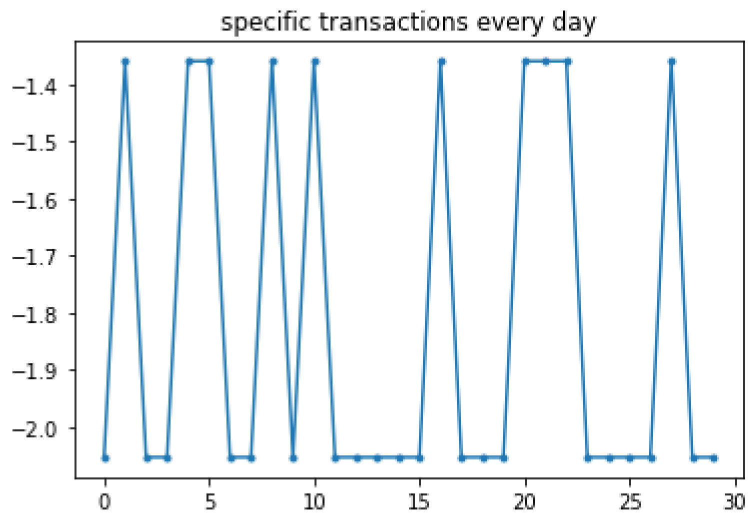
In [154]:

```
days_data = np.array(zip(kde_day.score_samples(days),range(1,31)))[:,0]
print days_data
plt.plot(days_data,marker='.')
plt.title("specific transactions every day")
```

```
[-2.05298845 -1.35984127 -2.05298845 -2.05298845 -1.35984127 -1.359841
27
 -2.05298845 -2.05298845 -1.35984127 -2.05298845 -1.35984127 -2.052988
45
 -2.05298845 -2.05298845 -2.05298845 -2.05298845 -1.35984127 -2.052988
45
 -2.05298845 -2.05298845 -1.35984127 -1.35984127 -1.35984127 -2.052988
45
 -2.05298845 -2.05298845 -2.05298845 -1.35984127 -2.05298845 -2.052988
45]
```

Out[154]:

```
Text(0.5,1,u'specific transactions every day')
```



Discussion regarding parameteres: The parameter used in KDE is bandwidth h, which is the doniminator in the kernel, influencinf sensiticity. "Mathematically, a kernel is a positive function K(x;h) which is controlled by the bandwidth parameter h. Given this kernel form, the density estimate at a point y within a group of points $x_i; i = 1 \cdots N$is given by:"

$$\rho_K(y) = \sum_{i=1}^{N} K((y - x_i)/h)$$

Therefore, we see that it influences the bias-variance tradoff: small bandwidth will lead to an unsmooth density distribution, with higher variance (and therefore bigger range that is more likely to include potential outcome), and higher h will lead to higher bias and smooth density distribution.

Part Two - fiction poetry:

In [153]:

```
#week10_assignment_topic_extraction.py

from collections import OrderedDict
import pandas as pd
from nltk.corpus import stopwords
import string
import re
import gensim
import numpy as np
from nltk.corpus import stopwords
from time import time

print "================================================== Data Prep: ========


headings = ['CHAPTER I', 'CHAPTER II',
            'CHAPTER III', 'CHAPTER IV', 'CHAPTER V',
            'CHAPTER VI', 'CHAPTER VII', 'CHAPTER VIII',
            'CHAPTER IX', 'CHAPTER X', 'CHAPTER XI',
            'CHAPTER XII', 'PREFACE', 'NOTE']

terminate_code = "THE END"

path_to_book = "ALICE.txt"

def extract_chapters(path_to_book):

    with open(path_to_book, 'r') as book:
        t = book.readlines()

    # We're going to put all the
    # sections/chapters in a dictionary
    chapter_dict = OrderedDict()

    # Initialize empty sections
    chapter_text = ''
    chapter_name = None

    for line in t:

        if not chapter_name and any([heading in line for heading in headings]):
            chapter_name = line.replace('\r\n', '')

        if chapter_name:

            # Populate section string with line
            if not any([heading in line for heading in headings]):
                chapter_text += line

            # If a heading line, we've hit a new
            # chapter/section; throw the text string into
            # the dictionary and start a new section
            if any([heading in line for heading in headings]):
                chapter_dict[chapter_name] = [chapter_text]
                chapter_name = line.replace('\r\n', '')
                chapter_text = ''

            # If we hit the end of the book,
            # throw everything into last chapter
```

```python
            if terminate_code in line:

                chapter_dict[chapter_name] = [chapter_text]

    # Make dataframe to store chapters
    df = pd.DataFrame.from_dict(chapter_dict, orient='index')
    df.columns = ['raw_text']

    return df

'''Clean up the text!'''

df = extract_chapters(path_to_book)

# Establish stopwords
sw = set(stopwords.words('english'))

# Remove punctuation & make lowercase
df['no_punctuation'] = df.raw_text.map(
    lambda x: x.translate(None, string.punctuation).lower())

# Remove stopwords
df['no_stopwords'] = df.no_punctuation.map(
    lambda x: [word for word in x.split() if word not in sw])

df['no_stopwords'] = df.no_stopwords.map(
    lambda x: " ".join(x))

print "============================================ Top Words Extraction : ====

n_components = 10
n_samples = 1000
n_features = 1000
n_top_words = 20

def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
    print()

print "============================================ TF - for LDA : ============

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# Use tf (raw term count) features for LDA.
print("Extracting tf features for LDA...")
tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2,
                                max_features=n_features,
                                stop_words='english')
t0 = time()
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
tf = tf_vectorizer.fit_transform(df["no_stopwords"])
print("done in %0.3fs." % (time() - t0))

print "============================================ LDA - Topic Extraction - Pe

from sklearn.decomposition import LatentDirichletAllocation

print("Fitting LDA models with tf features, "
```

```
      "n_samples=%d and n_features=%d..."
      % (n_samples, n_features))
lda = LatentDirichletAllocation(n_components=n_components, max_iter=5,
                                learning_method='online',
                                learning_offset=50.,
                                random_state=0)
t0 = time()
lda.fit(tf)
print("done in %0.3fs." % (time() - t0))

print("\nTopics in LDA model: These are the top 10 topics the model extracts fo
tf_feature_names = tf_vectorizer.get_feature_names()
print_top_words(lda, tf_feature_names, n_top_words)

print """Put together: \n
a hatter a king and a dormouse march with a hare a mouse and a queen.
The hatter didn stand the summer jury so he startled twinkling anxiously and sa
caterpillar came and the father felt bit mouth size took girls good inches.
Sharp question effected mushroom, the mouse, the rabbit ran to bring gloves soc
queen and duchess quite, but the cat say came to the turtle. The rabbit used la
The turtle gryphon mock soup beautiful(y) voice and won.
mouse came nad the cat quite. the caterpillar tell a thing and let voice of the
queen and king('s) voice came going, and soldiers anxiously quite white tone wi
A bottle wrote players small angrily eaglet sight unfolded march lives believe
king and the gryphon voice made the jury, queen,and the course and  the mouse l
"""

print "============================================ LDA - Topic Extraction, per

for index, row in enumerate(df["no_stopwords"]):
    lda = LatentDirichletAllocation(n_components=n_components, max_iter=5,
                                    learning_method='online',
                                    learning_offset=50.,
                                    random_state=0)
    print("\nTopics in LDA model: These are the top 10 topics the model extract
    tf = tf_vectorizer.fit_transform(row.split())
    lda.fit(tf)
    tf_feature_names = tf_vectorizer.get_feature_names()
    print_top_words(lda, tf_feature_names, n_top_words)

print "In this part, I worked with Skye Hersh, and she provided a very handy wa
```

```
======================================================================
================
============================================ TF - for LDA : ==========
======================================================================
========
Extracting tf features for LDA...
done in 0.041s.
============================================ LDA - Topic Extraction -
 Per Book : ==========================================================
==============================
Fitting LDA models with tf features, n_samples=1000 and n_features=100
0...
done in 0.851s.

Topics in LDA model: These are the top 10 topics the model extracts fo
r the whole book:
Topic #0: hatter king dormouse march hare mouse queen court ll rabbit

 white thing great quite say jury added voice course looking
Topic #1: sleep hatter shut didn stand summer jury sleepy march whiske
```

The code works in two different options, to examine different possibilites: running LDA over the whole book, and running over each chapter. The results of the first option are summarized into a short, interesting plot.

In [ ]:

```
 
```