Title: Data7202 A2 Report

Name: Xinqian Wang

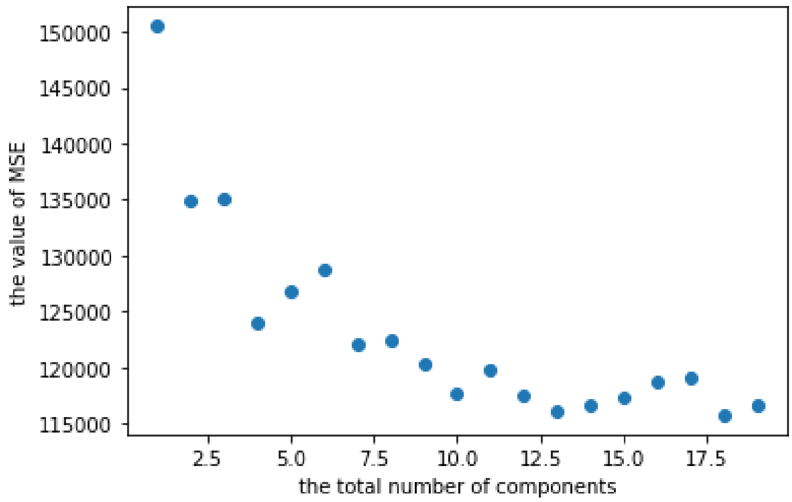
ID: s4565489

Tips: the code screenshot will be attached in the Appendix. For code script will also be attached in the folder.

- **1.**

Question: Apply Principal Component Regression (PCR) with all possible number of principal components. Using the 10-Fold Cross-Validation, plot the mean squared error as a function of the number of components and determine the optimal number of components.

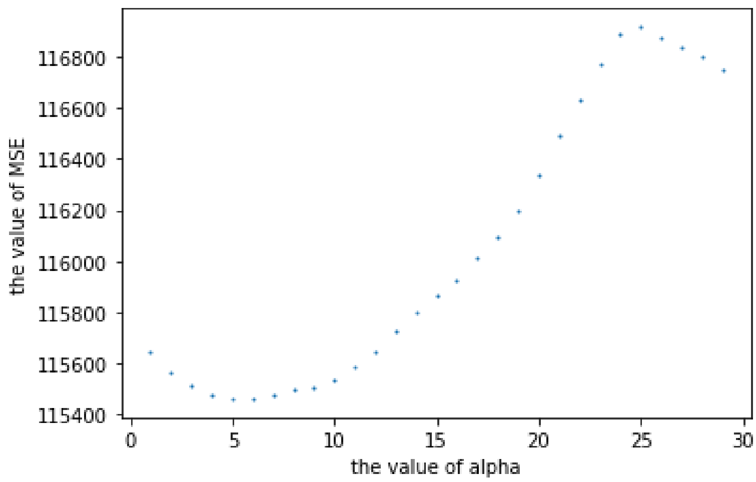
Answer:



The optimal number of components is 18.

Question: Apply the Lasso method and plot the 10-Fold Cross-Validation mean squared error as a function of λ. Determine the best λ and the corresponding mean squared error.

Answer:



The best λ should be the alpha equals to 5.

- 2.

Question: Construct a Poisson regression model and report the coefficients (for type, construction, operation, and months), and the corresponding 95% CIs.

Answer:

By using the code `model = sm.GLM(Y,X,family=sm.families.Poisson())`, we build a Poisson Regression Model. The coefficients for **type**, **construction**, **operation**, and **months** are below:

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                 damage   No. Observations:                   34
Model:                            GLM   Df Residuals:                       30
Model Family:                 Poisson   Df Model:                            3
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -145.96
Date:                Mon, 12 Apr 2021   Deviance:                       194.06
Time:                        13:30:54   Pearson chi2:                     178.
No. Iterations:                     6
Covariance Type:            nonrobust
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
type            -0.2237      0.048     -4.693      0.000      -0.317      -0.130
construction     0.3714      0.060      6.231      0.000       0.255       0.488
operation        0.7680      0.103      7.471      0.000       0.567       0.969
months        8.095e-05   2.84e-06     28.487      0.000    7.54e-05    8.65e-05
================================================================================
```

- 3.

Question: Compute the multiple regression of Time on Cases and Distance. State the fitted model, the estimated residual standard deviation, and the P-values for the overall model and each of the two predictors.

Answer:

```
Call:
lm(formula = Time ~ Cases + Distance, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7880 -0.6629  0.4364  1.1566  7.4197

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.341231   1.096730   2.135 0.044170 *
Cases       1.615907   0.170735   9.464 3.25e-09 ***
Distance    0.014385   0.003613   3.981 0.000631 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.259 on 22 degrees of freedom
Multiple R-squared:  0.9596,    Adjusted R-squared:  0.9559
F-statistic: 261.2 on 2 and 22 DF,  p-value: 4.687e-16
```

The Residual standard error $=\sqrt{\frac{\sum_{i=1}^{n}(Y_i-\widehat{Y_i})^2}{\mathrm{df}}}$ = 3.259 on 22 degrees of freedom.

The P-value of overall model is 4.687e-16.

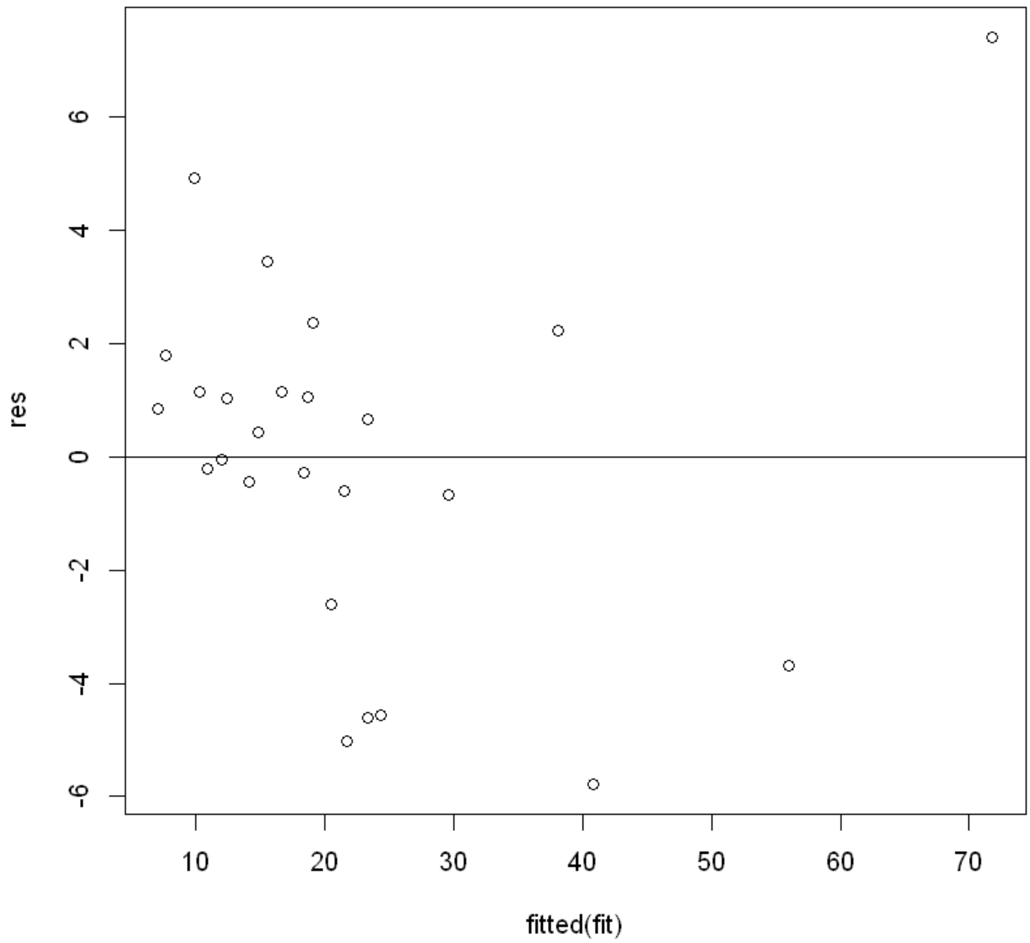
The P-value of the attribute Cases is 3.25e-09.

The P-value of the attribute Distance is 0.000631.

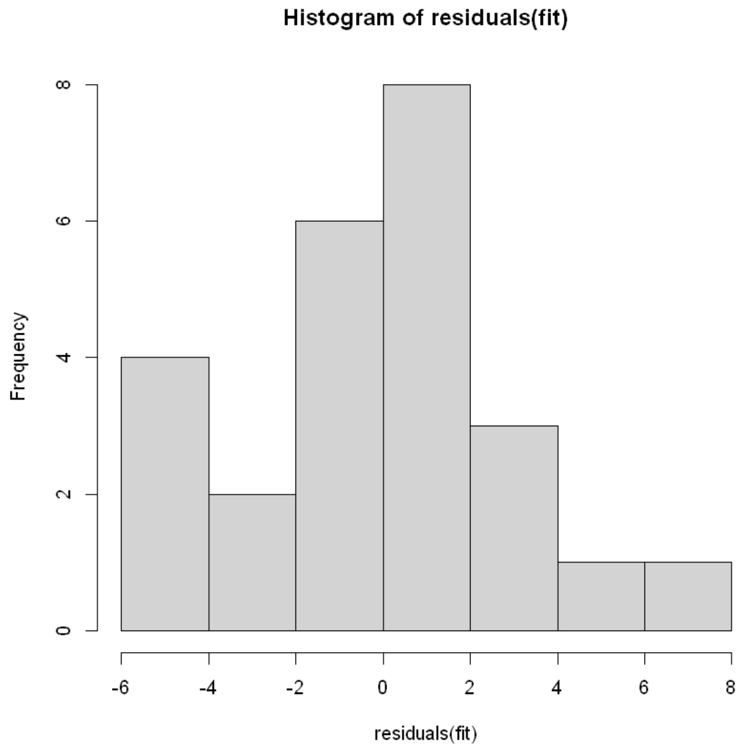
Question: Obtain residual plots and the histogram of the residuals. Comment on these.

Answer:

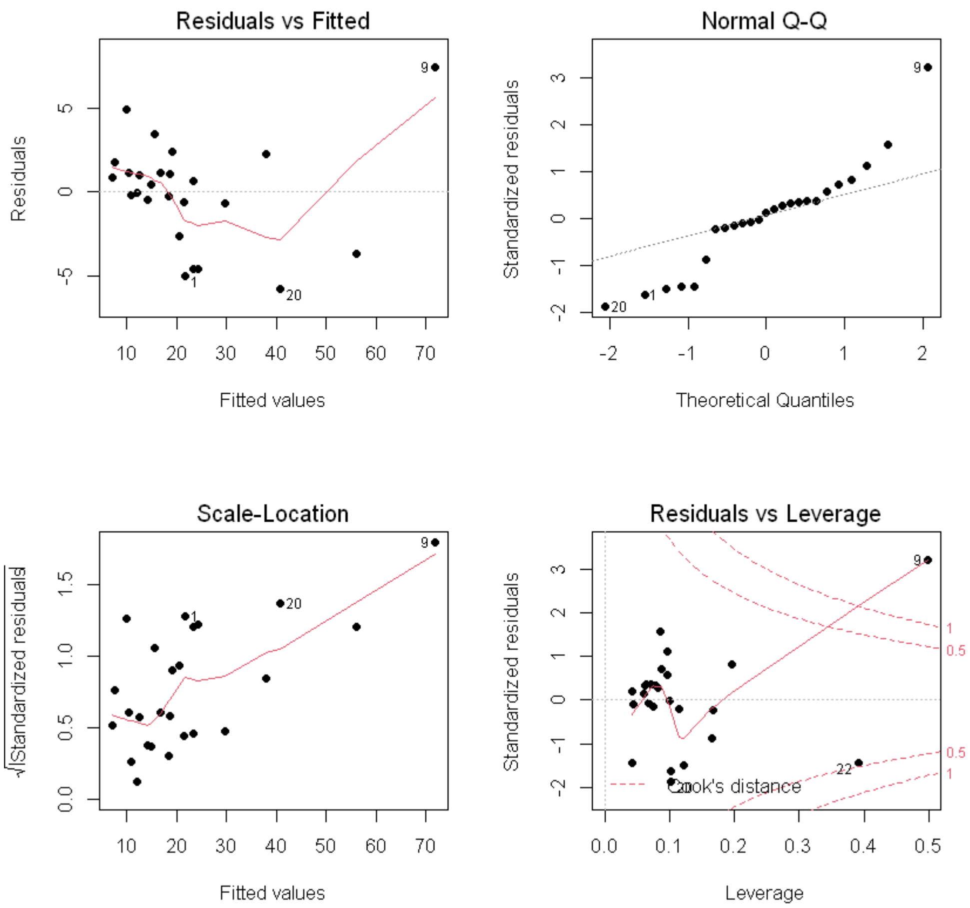
Residual plot is as below:



Residual histogram is as below:



Histogram of residuals(fit)
Frequency
residuals(fit)



Residuals vs Fitted
Residuals
Fitted values
Normal Q-Q
Standardized residuals
Theoretical Quantiles
Scale-Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized residuals
Cook's distance
Leverage

The **histogram** shows most of the residuals have the values range between -2 and 2.

The **Residuals vs Fitted plot** shows whether the residuals have non-linear patterns. The plot fails to show equally spread residuals around a horizontal line without distinct patterns, which indicates that the non-linear relationship was not explained by the model and was left out in the residuals.

The **QQ plot** is used to show if the residuals are normally distributed or not. The plot just tells that the residuals are probably not following a normal distribution.

The **Spread-Location plot** shows if residuals are spread equally along the ranges of predictors. The plot shows that most of the residuals are spread in the left side of the plot whereas only a few points distributed on the right. Especially the 9th point, it' s value seems not common as other points.

This **Residuals vs Leverage plot** helps us to find influential cases if any. Not all outliers are influential in linear regression analysis (whatever outliers mean). It is usually the case that a point is judged as an outlier if its cook's distance is greater than 0.5. In this case, the point 22 and point 9 would interest us to do further research, which also means they are more influential than any other points when the regression model is decided.

Question: There is an observation in this data set which is extremely influential according to Cook' s distance. Which observation is it? Display a Cook' s distance plot to determine the Cook' s distance of the next most influential observation.

Answer:

The 9-th point is the most influential point, according to the Influence table below and the plots above. The Cook' s distance plot is placed on the former plot. The next most influential observation would be the 22-ed point.

```
Influence measures of
	 lm(formula = Time ~ Cases + Distance, data = data) :

     dfb.1_ dfb.Cass dfb.Dstn   dffit cov.r   cook.d    hat inf
1  -0.18727  0.41131 -0.43486 -0.5709 0.871 1.00e-01 0.1018
2   0.08979 -0.04776  0.01441  0.0986 1.215 3.38e-03 0.0707
3  -0.00352  0.00395 -0.00285 -0.0052 1.276 9.46e-06 0.0987
4   0.45196  0.08828 -0.27337  0.5008 0.876 7.76e-02 0.0854
5  -0.03167 -0.01330  0.02424 -0.0395 1.240 5.43e-04 0.0750
6  -0.01468  0.00179  0.00108 -0.0188 1.200 1.23e-04 0.0429
7   0.07807 -0.02228 -0.01102  0.0790 1.240 2.17e-03 0.0818
8   0.07120  0.03338 -0.05382  0.0938 1.206 3.05e-03 0.0637
9  -2.57574  0.92874  1.50755  4.2961 0.342 3.42e+00 0.4983   *
10  0.10792 -0.33816  0.34133  0.3987 1.305 5.38e-02 0.1963
11 -0.03427  0.09253 -0.00269  0.2180 1.172 1.62e-02 0.0861
12 -0.03027 -0.04867  0.05397 -0.0677 1.291 1.60e-03 0.1137
13  0.07237 -0.03562  0.01134  0.0813 1.207 2.29e-03 0.0611
14  0.04952 -0.06709  0.06182  0.0974 1.228 3.29e-03 0.0782
15  0.02228 -0.00479  0.00684  0.0426 1.192 6.32e-04 0.0411
16 -0.00269  0.06442 -0.08419 -0.0972 1.369 3.29e-03 0.1659
17  0.02886  0.00649 -0.01570  0.0339 1.219 4.01e-04 0.0594
18  0.24856  0.18973 -0.27243  0.3653 1.069 4.40e-02 0.0963
19  0.17256  0.02357 -0.09897  0.1862 1.215 1.19e-02 0.0964
20  0.16804 -0.21500 -0.09292 -0.6718 0.760 1.32e-01 0.1017
21 -0.16193 -0.29718  0.33641 -0.3885 1.238 5.09e-02 0.1653
22  0.39857 -1.02541  0.57314 -1.1950 1.398 4.51e-01 0.3916   *
23 -0.15985  0.03729 -0.05265 -0.3075 0.890 2.99e-02 0.0413
24 -0.11972  0.40462 -0.46545 -0.5711 0.948 1.02e-01 0.1206
25 -0.01682  0.00085  0.00559 -0.0176 1.231 1.08e-04 0.0666
```

- 4.

Question: Derive the posterior distribution of θ.

Answer:

Firstly, calculating the likelihood:

$$p(y \mid \theta) = \frac{1}{\theta}$$

Secondly, calculating the prior:

$$\mathbf{p}(\theta \mid \alpha, x_m) = \begin{cases} \dfrac{\alpha x_m^{\alpha}}{\theta^{\alpha+1}} & \theta \geqslant x_m \\ 0 & \theta < x_m \end{cases}$$

Thirdly, calculating the posterior and prove it proportional to a Pareto distribution:

$$
\begin{aligned}
P(\theta \mid Y) &= \frac{P(Y \mid \theta) \cdot P(\theta)}{P(Y)} \\
&\propto P(Y \mid \theta) \cdot P(\theta) \\
&= \left(\prod_{i=1}^{n} \mathbf{p}(y_i \mid \theta) \right) \cdot \mathbf{p}\left(\theta \mid \alpha, x_m\right) \\
&= \frac{1}{\theta^n} \cdot \mathbf{p}\left(\theta \mid \alpha, x_m\right) \\
&= \begin{cases} \frac{\alpha x_m^{\alpha}}{\theta^{(n+\alpha)+1}} & \theta \geqslant x_m \\ 0 & \theta < x_m \end{cases} \\
&= \frac{\alpha x_m^{-n}}{n+\alpha} \cdot \begin{cases} \frac{(n+\alpha) x_m^{(\alpha+n)}}{\theta^{(n+\alpha)+1}} & \theta \geqslant x_m \\ 0 & \theta < x_m \end{cases} \\
&\propto \mathbf{Pareto}\left(\theta \mid (n+\alpha), x_m\right)
\end{aligned}
$$

- 5.

Question: Find the conditional pdf of X given Y = y, and the conditional pdf of Y given X = x.

Answer:

For any y such that $f_Y(y) > 0$, the conditional pdf of X given that $Y = y$ is the function of x denoted by $f(x \mid y)$ and defined by

$f(y \mid x) = \frac{f(x,y)}{f_X(x)}$

$f(x \mid y) = \frac{f(x,y)}{f_Y(y)}$

$\forall\ x \geqslant 0,\ y \geqslant 0$

$$
\begin{aligned}
f_X(x) &= \int_0^{+\infty} f(x, y) dy \\
&= \int_0^{+\infty} ce^{-(xy+x+y)} dy \\
&= \int_0^{+\infty} c \cdot e^{-x} \cdot e^{-(x+1)y} dy \\
&= c \cdot e^{-x} \cdot \frac{1}{-(1+x)} \cdot \lim_{y \to +\infty} (e^{-(1+x)y} - 1) \\
&= \frac{c \cdot e^{-x}}{1+x}
\end{aligned}
$$

Similarly, marginal of Y:

$$f_Y(y) = \frac{c \cdot e^{-y}}{1+y}$$

$f(y \mid x) = \frac{f(x,y)}{f_X(x)} = (1+x)e^{-y(1+x)} = Exp(y; \lambda = (1+x))$

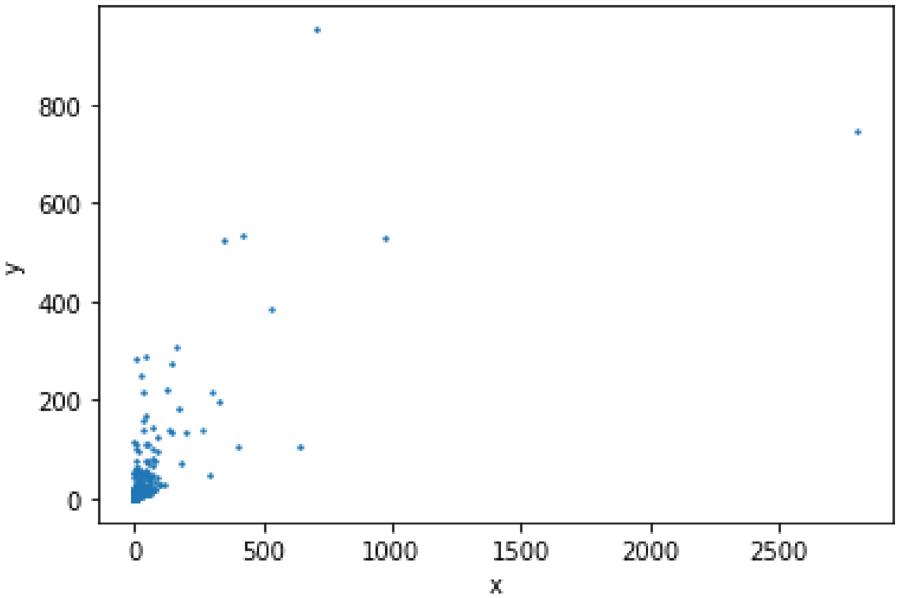
$f(x \mid y) = \frac{f(x,y)}{f_Y(y)} = (1+y)e^{-x(1+y)} = Exp(x; \lambda = (1+y))$

Conclusion, for given Y=y: $X \sim Exp(1+y)$
for given X=x: $Y \sim Exp(1+x)$

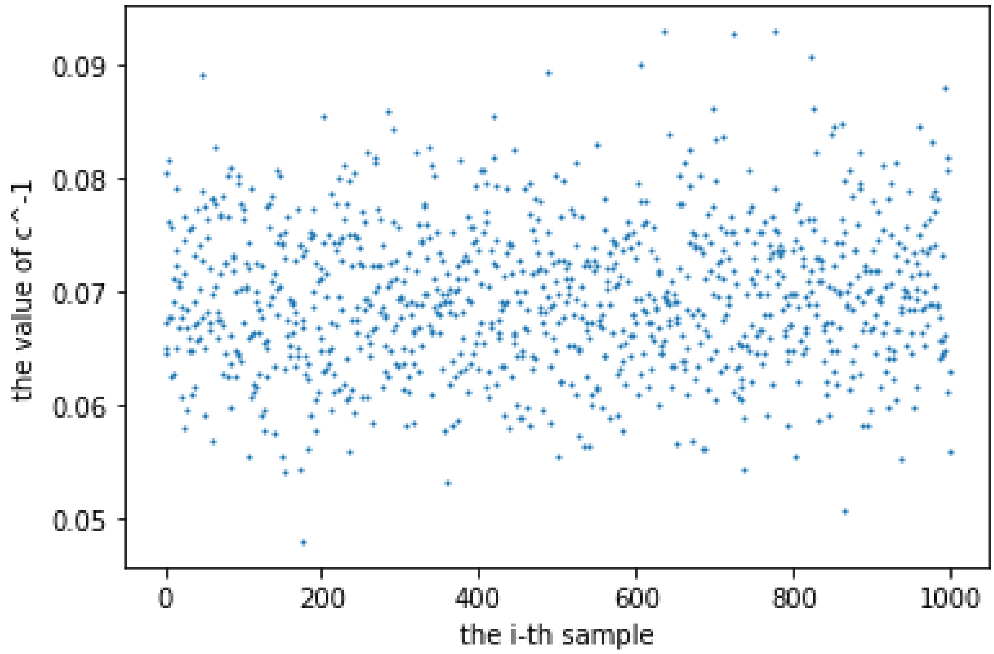
Question: Write working Python code that implements the Gibbs sampler and outputs 1000 points that are approximately distributed according to f.

Answer:

Attached in the code. The scatter plot as below:



I can also calculate the approximation of the constant c since it is hard to be calculated from the equation of $\int_0^{+\infty} \int_0^{+\infty} f(x, y)\ dydx = 1$. By calculating the sum of the 1000 samples and divided by 1000, we can get a good approximation of c. Repeating it with 1000 times, we can get the range of value c^{-1}. I also calculated the mean value of c^{-1}, which is around **0.069578**. I draw a scatter plot with the value of 1000 number of c^{-1} below:



Appendix

- 1.

```
In [1]: # Package implementation
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
In [2]: df = pd.read_csv("Hitters.csv")
#  Change Data Type
labelencoder = LabelEncoder()
df['League'] = labelencoder.fit_transform(df['League'])
df['Division'] = labelencoder.fit_transform(df['Division'])
df['NewLeague'] = labelencoder.fit_transform(df['NewLeague'])


#Generate X, Y
X_cols = [i for i in df.columns if i not in ['Salary']]
Y_cols = [i for i in df.columns if i in ['Salary']]
X = df[X_cols]
Y = df[Y_cols]
```

Q1_a

In [3]:
```
def PCR(X,Y,N_Cols,model):
    #PCA
    X = PCA(n_components=N_Cols).fit_transform(X)

    X = pd.DataFrame(X)

    kf = KFold(n_splits=10)
    kf.get_n_splits(X)
    MSE = []

    for train_index, test_index in kf.split(X):
        #print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
        y_train, y_test = Y.iloc[train_index], Y.iloc[test_index]
        # fit the model
        model.fit(X_train,y_train)
        # predict
        y_pred = model.predict(X_test)
        loss = mean_squared_error(y_test,y_pred)
        MSE.append(loss)
    return np.mean(MSE)
```

In [4]:
```
model = LinearRegression()
PCR(X,Y,19,model)
Error_List = []
Component_List = [i for i in range(1,20)]
for i in range(1,20):
    Error_List.append(PCR(X,Y,i,model))
```

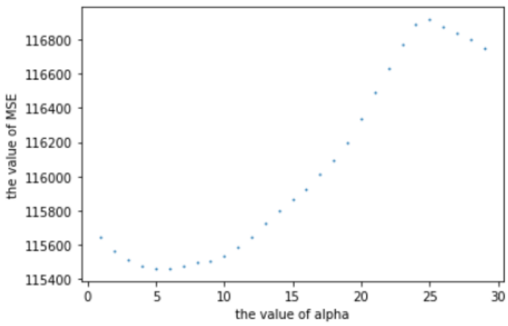
In [5]:
```
plt.xlabel("the total number of components")
plt.ylabel("the value of MSE")
plt.scatter([i for i in range(1,20)], Error_List)
```

Q1_b

In [8]:
```
lasso = linear_model.Lasso(fit_intercept=True, alpha=5)
print(PCR(X,Y,18,lasso))
Error_List = []
Component_List = [i for i in range(1,20)]
for i in range(1,30):
    Error_List.append(PCR(X,Y,18,linear_model.Lasso(fit_intercept=True, alpha=i)))
plt.xlabel("the value of alpha")
plt.ylabel("the value of MSE")
plt.scatter([i for i in range(1,30)], Error_List,s = 1)
```

```
115462.2995924692
```

Out[8]: <matplotlib.collections.PathCollection at 0x2c64d085e88>



- 2.

```
In [1]: # Package implementation
        import pandas as pd
        import numpy as np
        #from sklearn.preprocessing import OneHotEncoder
        #from sklearn.preprocessing import LabelEncoder

        import statsmodels.api as sm
        from statsmodels.formula.api import ols
        from statsmodels.formula.api import poisson
        import matplotlib.pyplot as plt
```

```
In [2]: df = pd.read_csv("ships.csv")
        '''
        enc = OneHotEncoder(handle_unknown='ignore')
        labelencoder = LabelEncoder()

        enc_df_League = pd.DataFrame(enc.fit_transform(df[['type']]).toarray())
        df['Type_A'] = labelencoder.fit_transform(enc_df_League[0])
        df['Type_B'] = labelencoder.fit_transform(enc_df_League[1])
        df['Type_C'] = labelencoder.fit_transform(enc_df_League[2])
        df['Type_D'] = labelencoder.fit_transform(enc_df_League[3])
        df['Type_E'] = labelencoder.fit_transform(enc_df_League[4])

        enc_df_League = pd.DataFrame(enc.fit_transform(df[['construction']]).toarray())
        df['Con_1'] = labelencoder.fit_transform(enc_df_League[0])
        df['Con_2'] = labelencoder.fit_transform(enc_df_League[1])
        df['Con_3'] = labelencoder.fit_transform(enc_df_League[2])
        df['Con_4'] = labelencoder.fit_transform(enc_df_League[3])
        '''

        #df = df.drop(['type', 'construction'], axis=1)
```

```
In [4]: Y = df["damage"]
        X = df.drop(['damage'], axis=1)
```

```
In [5]: model = sm.GLM(Y,X,family=sm.families.Poisson())
        results = model.fit()
        print(results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                 damage   No. Observations:                   34
Model:                            GLM   Df Residuals:                       30
Model Family:                 Poisson   Df Model:                            3
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -145.96
Date:                Mon, 12 Apr 2021   Deviance:                       194.06
Time:                        13:30:54   Pearson chi2:                     178.
No. Iterations:                     6
Covariance Type:            nonrobust
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
type            -0.2237      0.048     -4.693      0.000      -0.317      -0.130
construction     0.3714      0.060      6.231      0.000       0.255       0.488
operation        0.7680      0.103      7.471      0.000       0.567       0.969
months        8.095e-05   2.84e-06     28.487      0.000    7.54e-05    8.65e-05
================================================================================
```

- 3.

Python Kernel

Python Kernel (Please execute the R-Kernel model below:

In [48]:
```
# Package implementation
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.graphics.gofplots import ProbPlot
import matplotlib.pyplot as plt
import pylab
import seaborn as sns
```

In [49]:
```
df = pd.read_csv("softdrink.csv")
df.insert(df.shape[1], 'Interept', 1)
```

In [50]:
```
Y = df["Time"]
X = df.drop(['Time'], axis=1)
```

In [51]:
```
multi_lin_reg = sm.OLS(Y, X).fit()
print(multi_lin_reg.summary())
```

In [52]:
```
#residual standard error/residual standard deviation
a = abs(Y-multi_lin_reg.predict(X)).values
a = a**2
(sum(a)/22)**0.5
```

Out[52]: 3.2594734475800964

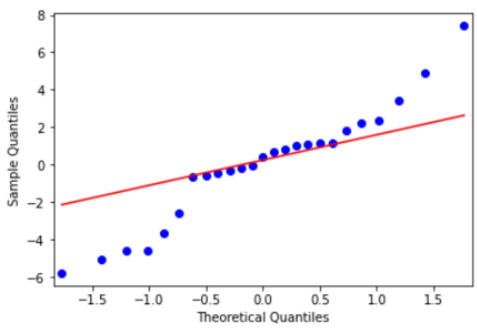
In [53]:
```
'''
a = abs(Y-multi_lin_reg.predict(X)).values
print(np.std(a, axis=None, dtype=None, out=None, ddof=3, keepdims=np._NoValue))
print((sum((a-np.mean(a))**2)/(np.shape(a)[0]-3))**(0.5))
'''
```

Out[53]:
```
'\na = abs(Y-multi_lin_reg.predict(X)).values\nprint(np.std(a, axis=None, dtype=
((a-np.mean(a))**2)/(np.shape(a)[0]-3))**(0.5))\n'
```

In [54]:
```
# pvalues for parameters
multi_lin_reg.pvalues
```

Out[54]:
```
Cases       3.254932e-09
Distance    6.312469e-04
Interept    4.417012e-02
dtype: float64
```

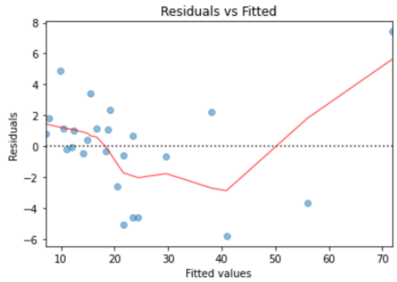
In [55]:
```
sm.qqplot(Y-multi_lin_reg.predict(X).values, line='q')
pylab.show()
```



In [56]:
```
# model values
model_fitted_y = multi_lin_reg.fittedvalues
# model residuals
model_residuals = multi_lin_reg.resid
# normalized residuals
model_norm_residuals = multi_lin_reg.get_influence().resid_studentized_internal
# absolute squared normalized residuals
model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
# absolute residuals
model_abs_resid = np.abs(model_residuals)
# leverage, from statsmodels internals
model_leverage = multi_lin_reg.get_influence().hat_matrix_diag
# cook's distance, from statsmodels internals
model_cooks = multi_lin_reg.get_influence().cooks_distance[0]
```

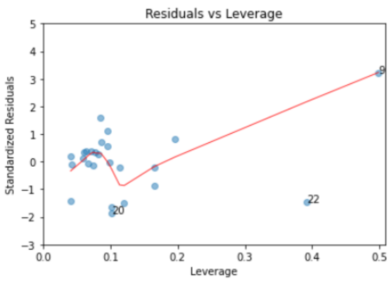
In [57]:
```
plot_lm_1 = plt.figure()
plot_lm_1.axes[0] = sns.residplot(model_fitted_y, Y, data=df,lowess=True,
                                  scatter_kws={'alpha': 0.5},line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plot_lm_1.axes[0].set_title('Residuals vs Fitted')
plot_lm_1.axes[0].set_xlabel('Fitted values')
plot_lm_1.axes[0].set_ylabel('Residuals');
```

```
c:\users\andy\appdata\local\programs\python\python37\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [59]:
```
plot_lm_4 = plt.figure()
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
sns.regplot(model_leverage, model_norm_residuals,scatter=False,ci=False,lowess=True,line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
plot_lm_4.axes[0].set_xlim(0, max(model_leverage)+0.01)
plot_lm_4.axes[0].set_ylim(-3, 5)
plot_lm_4.axes[0].set_title('Residuals vs Leverage')
plot_lm_4.axes[0].set_xlabel('Leverage')
plot_lm_4.axes[0].set_ylabel('Standardized Residuals');

# annotations
leverage_top_3 = np.flip(np.argsort(model_cooks), 0)[:3]
for i in leverage_top_3:
    plot_lm_4.axes[0].annotate(i+1, xy=(model_leverage[i],model_norm_residuals[i]));
```

```
c:\users\andy\appdata\local\programs\python\python37\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



R Kernel

R Kernel

In [1]:
```
# read data
data = read.csv("softdrink.csv")
data
```

A data.frame: 25 × 3

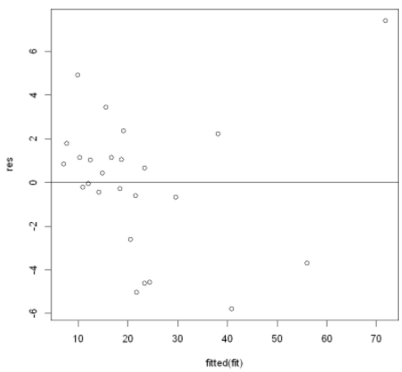
Time	Cases	Distance
<dbl>	<int>	<int>
16.68	7	560
11.50	3	220
12.03	3	340

```
In [2]: fit = lm(Time~Cases+Distance,data=data)
        summary(fit)
```

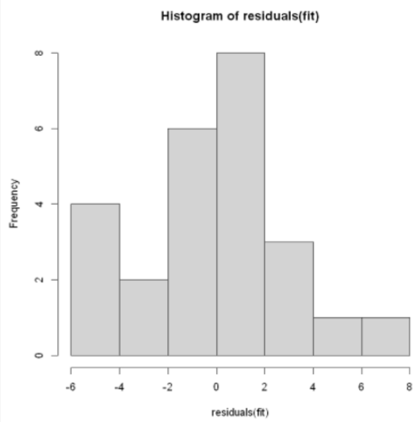
```
Call:
lm(formula = Time ~ Cases + Distance, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7880 -0.6629  0.4364  1.1566  7.4197

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.341231   1.096730   2.135 0.044170 *
Cases       1.615907   0.170735   9.464 3.25e-09 ***
Distance    0.014385   0.003613   3.981 0.000631 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.259 on 22 degrees of freedom
Multiple R-squared:  0.9596,    Adjusted R-squared:  0.9559
F-statistic: 261.2 on 2 and 22 DF,  p-value: 4.687e-16
```

$$\text{Residual standard error} = \sqrt{\frac{\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}{\text{df}}}$$

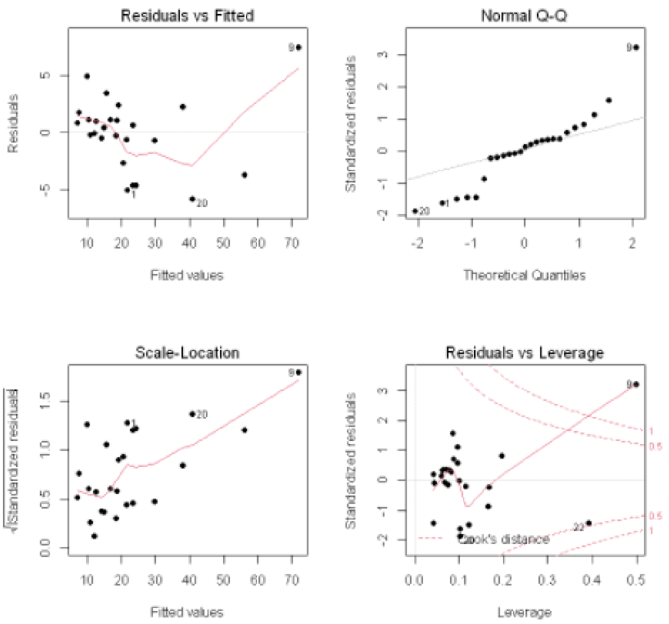
```
In [6]: res = resid(fit)
        plot(fitted(fit), res)
        abline(0,0)
```




```
In [3]: hist(residuals(fit))
```



In [4]:
```r
par(mfrow=c(2,2))
plot(fit,pch=16)
```



In [5]:
```r
influence.measures(fit)
```

```
Influence measures of
         lm(formula = Time ~ Cases + Distance, data = data) :

     dfb.1_ dfb.Cass dfb.Dstn   dffit cov.r   cook.d    hat inf
1  -0.18727  0.41131 -0.43486 -0.5709 0.871 1.00e-01 0.1018
2   0.08979 -0.04776  0.01441  0.0986 1.215 3.38e-03 0.0707
3  -0.00352  0.00395 -0.00285 -0.0052 1.276 9.46e-06 0.0987
4   0.45196  0.08828 -0.27337  0.5008 0.876 7.76e-02 0.0854
5  -0.03167 -0.01330  0.02424 -0.0395 1.240 5.43e-04 0.0750
6  -0.01468  0.00179  0.00108 -0.0188 1.200 1.23e-04 0.0429
7   0.07807 -0.02228 -0.01102  0.0790 1.240 2.17e-03 0.0818
8   0.07120  0.03338 -0.05382  0.0938 1.206 3.05e-03 0.0637
9  -2.57574  0.92874  1.50755  4.2961 0.342 3.42e+00 0.4983   *
10  0.10792 -0.33816  0.34133  0.3987 1.305 5.38e-02 0.1963
11 -0.03427  0.09253 -0.00269  0.2180 1.172 1.62e-02 0.0861
12 -0.03027 -0.04867  0.05397 -0.0677 1.291 1.60e-03 0.1137
13  0.07237 -0.03562  0.01134  0.0813 1.207 2.29e-03 0.0611
14  0.04952 -0.06709  0.06182  0.0974 1.228 3.29e-03 0.0782
15  0.02228 -0.00479  0.00684  0.0426 1.192 6.32e-04 0.0411
16 -0.00269  0.06442 -0.08419 -0.0972 1.369 3.29e-03 0.1659
17  0.02886  0.00649 -0.01570  0.0339 1.219 4.01e-04 0.0594
18  0.24856  0.18973 -0.27243  0.3653 1.069 4.40e-02 0.0963
19  0.17256  0.02357 -0.09897  0.1862 1.215 1.19e-02 0.0964
20  0.16804 -0.21500 -0.09292 -0.6718 0.760 1.32e-01 0.1017
21 -0.16193 -0.29718  0.33641 -0.3885 1.238 5.09e-02 0.1653
22  0.39857 -1.02541  0.57314 -1.1950 1.398 4.51e-01 0.3916   *
23 -0.15985  0.03729 -0.05265 -0.3075 0.890 2.99e-02 0.0413
24 -0.11972  0.40462 -0.46545 -0.5711 0.948 1.02e-01 0.1206
25 -0.01682  0.00085  0.00559 -0.0176 1.231 1.08e-04 0.0666
```

- 4.

None

- 5.

In [1]:
```
#Import libraries
from scipy.stats import expon
import math
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#Set the seed
np.random.seed(44)
```

In [3]:
```
# initial values
x = 1
```

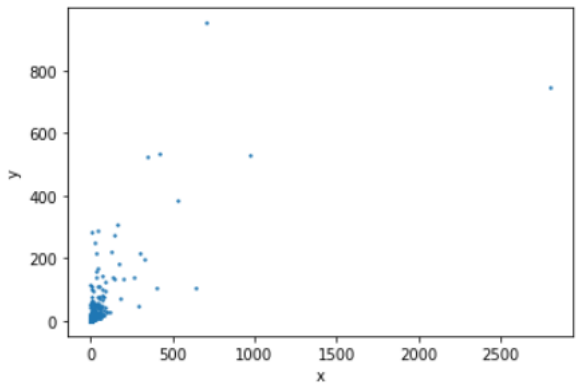
In [4]:
```
# simulation parameters
N = 1000
burnin = 300
```

In [5]:
```
#Using a list store the x_i and y_i
X_Sample = []
Y_Sample = []
```

In [6]:
```
for i in range(N+burnin):
    #Sampling y for given x
    y = float(np.random.exponential(1+x, 1))
    x = float(np.random.exponential(1+y, 1))

    #Upgrading
    if i >= burnin:
        #Store the data
        X_Sample.append(x)
        Y_Sample.append(y)
```

In [7]:
```
synth_plot = plt.scatter(X_Sample, Y_Sample,s=2)
plt.xlabel("x")
plt.ylabel("y")
```

Out[7]: Text(0, 0.5, 'y')



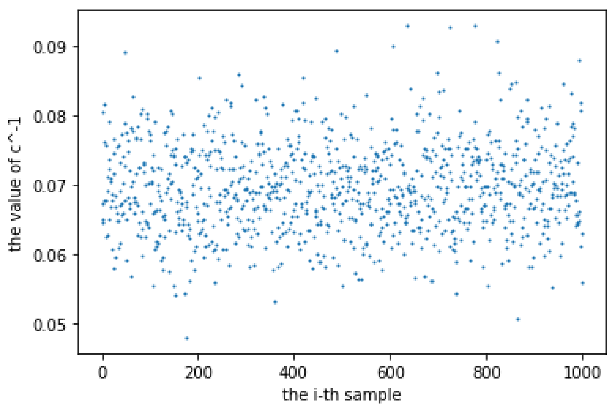
In [8]:
```
sum = 0
for i in range(N):
    x = X_Sample[i]
    y = Y_Sample[i]
    k = x+y+x*y
    try:
        s = 1/(math.exp(k))
    except OverflowError:
        s = 0
    sum = sum + s
print(sum/1000)
```

0.06896669414414609

```
In [9]: def Gibbs_Sampling(N, burnin, initial_value,seed=None):
            if seed!=None:
                np.random.seed(seed)

            X_Sample = []
            Y_Sample = []
            x = initial_value
            for i in range(N+burnin):
                #Sampling y for given x
                y = float(np.random.exponential(1+x, 1))
                x = float(np.random.exponential(1+y, 1))

                #Upgrading
                if i >= burnin:
                    #Store the data
                    X_Sample.append(x)
                    Y_Sample.append(y)
            sum = 0
            for i in range(N):
                x = X_Sample[i]
                y = Y_Sample[i]
                k = x+y+x*y
                try:
                    s = 1/(math.exp(k))
                except OverflowError:
                    s = 0
                sum = sum + s
            return sum/N
```

```
In [10]: C = []
         N = 1000
         burnin = 100
         initial_value = 10
         for i in range(N):
             C.append(Gibbs_Sampling(N, burnin, initial_value))
         synth_plot = plt.scatter([i for i in range(N)], C, s=1)
         plt.xlabel("the i-th sample")
         plt.ylabel("the value of c^-1")
         print("Average is: ",np.mean(C))
```

```
Average is:  0.06960987528553901
```



Thank you for reading my report.