9 : Estimating Insurance Claim Amounts : Problem Statement: Estimating Insurance Claim Amounts Project Description: Develop a regression model to estimate the claim amounts for insurance policies based on customer profiles, policy details, and historical claim data. Domain: Insurance Dataset Link:https://www.kaggle.com/competitions/allstate-claims-severity/data?select=train.csv (https://www.kaggle.com/competitions/allstate-claims-severity/data?select=train.csv)

Regression:Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as temperature, age, salary, price, etc.

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, "Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum." The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

```
Prediction of rain using temperature and other factors
Determining Market trends
Prediction of road accidents due to rash driving.
```

In [3]:

```
# importing Libraries
# importing Pandas Library as pd
import pandas as pd

# importing Numpy Library as np
import numpy as np

# importing matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# imporing seaborn as sns
import seaborn as sns

```

In [4]:
```
# Loading the dataset using pandas module and assign it as df
df_test = pd.read_csv('test.csv')


# Printing the dataset
df_test
```

Out[4]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont5	cont6
0	4	A	B	A	A	A	A	A	A	B	...	0.281143	0.466591
1	6	A	B	A	B	A	A	A	A	B	...	0.836443	0.482425
2	9	A	B	A	B	B	A	B	A	B	...	0.718531	0.212308
3	12	A	A	A	A	B	A	A	A	A	...	0.397069	0.369930
4	15	B	A	A	A	A	B	A	A	A	...	0.302678	0.398862
...	...	...	...	...	...	...	...	...	...	...	...	...	...
125541	587617	A	A	A	B	A	A	A	A	A	...	0.281143	0.438917
125542	587621	A	A	A	A	B	B	A	B	A	...	0.674529	0.346948
125543	587627	B	B	A	A	B	A	A	A	B	...	0.794794	0.808958
125544	587629	A	A	A	A	A	B	A	B	A	...	0.302678	0.372125
125545	587634	A	B	A	A	A	A	A	A	B	...	0.413817	0.221699

125546 rows × 131 columns

In [5]:
```
# Loading the dataset using pandas module and assign it as df
df_train = pd.read_csv('train.csv')

# Printing the dataset
df_train
```

Out[5]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	0.335060
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	0.436585
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	0.315545
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	0.391128
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	0.247408
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188313	587620	A	B	A	A	A	A	A	A	B	...	0.242437	0.289949
188314	587624	A	A	A	A	A	B	A	A	A	...	0.334270	0.382000
188315	587630	A	B	A	A	A	A	A	B	B	...	0.345883	0.370534
188316	587632	A	B	A	A	A	A	A	A	B	...	0.704364	0.562866
188317	587633	B	A	A	B	A	A	A	A	A	...	0.844563	0.533048

188318 rows × 132 columns

In [6]:
```
df = pd.concat([df_train, df_test], join='outer', axis=0)
print(df)
```

```
            id cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9  ...
cont6  \
0            1    A    B    A    B    A    A    A    A    B  ...
0.718367
1            2    A    B    A    A    A    A    A    A    B  ...
0.438917
2            5    A    B    A    A    B    A    A    A    B  ...
0.289648
3           10    B    B    A    B    A    A    A    A    B  ...
0.440945
4           11    A    B    A    B    A    A    A    A    B  ...
0.178193
...        ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
...
125541  587617    A    A    A    B    A    A    A    A    A  ...
0.438917
125542  587621    A    A    A    A    B    B    A    B    A  ...
0.346948
125543  587627    B    B    A    A    B    A    A    A    B  ...
0.808958
125544  587629    A    A    A    A    A    B    A    B    A  ...
0.372125
125545  587634    A    B    A    A    A    A    A    A    B  ...
0.221699

           cont7    cont8    cont9   cont10    cont11    cont12
cont13  \
0       0.335060  0.30260  0.67135  0.83510  0.569745  0.594646  0.
822493
1       0.436585  0.60087  0.35127  0.43919  0.338312  0.366307  0.
611431
2       0.315545  0.27320  0.26076  0.32446  0.381398  0.373424  0.
195709
3       0.391128  0.31796  0.32128  0.44467  0.327915  0.321570  0.
605077
4       0.247408  0.24564  0.22089  0.21230  0.204687  0.202213  0.
246011
...          ...      ...      ...      ...       ...       ...
...
125541  0.815941  0.39455  0.48740  0.40666  0.550529  0.538473  0.
298734
125542  0.424968  0.47669  0.25753  0.26894  0.324486  0.352251  0.
490001
125543  0.511502  0.72299  0.94438  0.83510  0.933174  0.926619  0.
848129
125544  0.388545  0.31796  0.32128  0.36974  0.307628  0.301921  0.
608259
125545  0.242044  0.25461  0.31399  0.25183  0.245410  0.241676  0.
287682

          cont14     loss
0       0.714843  2213.18
1       0.304496  1283.60
2       0.774425  3005.09
3       0.602642   939.85
4       0.432606  2763.85
...          ...      ...
125541  0.345946      NaN
125542  0.290576      NaN
125543  0.808125      NaN
```

```
125544  0.361542      NaN
125545  0.220323      NaN
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 313864 entries, 0 to 125545
Columns: 132 entries, id to loss
dtypes: float64(15), int64(1), object(116)
memory usage: 318.5+ MB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
id             0
cat1           0
cat2           0
cat3           0
cat4           0
           ...
cont11         0
cont12         0
cont13         0
cont14         0
loss      125546
Length: 132, dtype: int64
```

In [8]:
```
df.dropna(inplace = True)
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
id        0
cat1      0
cat2      0
cat3      0
cat4      0
         ..
cont11    0
cont12    0
cont13    0
cont14    0
loss      0
Length: 132, dtype: int64
```

In [10]:
```
print(df.size)
print(df.shape)
```

```
24857976
(188318, 132)
```

Information about the data

In [11]:
```
df.head()
```

Out[11]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	0.335060	0.30260
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	0.436585	0.60087

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	0.315545	0.27320
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	0.391128	0.31796
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	0.247408	0.24564

In [12]:
```
df.tail()
```

Out[12]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7
188313	587620	A	B	A	A	A	A	A	A	B	...	0.242437	0.289949
188314	587624	A	A	A	A	A	B	A	A	A	...	0.334270	0.382000
188315	587630	A	B	A	A	A	A	A	B	B	...	0.345883	0.370534
188316	587632	A	B	A	A	A	A	A	A	B	...	0.704364	0.562866
188317	587633	B	A	A	B	A	A	A	A	A	...	0.844563	0.533048

5 rows × 132 columns

In [14]:
```
# Summary stats for Numerical Column:
df.describe()
```

Out[14]:

	id	cont1	cont2	cont3	cont4	
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.0
mean	294135.982561	0.493861	0.507188	0.498918	0.491812	0.4
std	169336.084867	0.187640	0.207202	0.202105	0.211292	0.2
min	1.000000	0.000016	0.001149	0.002634	0.176921	0.2
25%	147748.250000	0.346090	0.358319	0.336963	0.327354	0.2
50%	294539.500000	0.475784	0.555782	0.527991	0.452887	0.4
75%	440680.500000	0.623912	0.681761	0.634224	0.652072	0.6
max	587633.000000	0.984975	0.862654	0.944251	0.954297	0.9

In [15]:
```
# Summary stats for Categorical Column:
from warnings import filterwarnings
filterwarnings('ignore')
df.describe(include = [np.object])
```

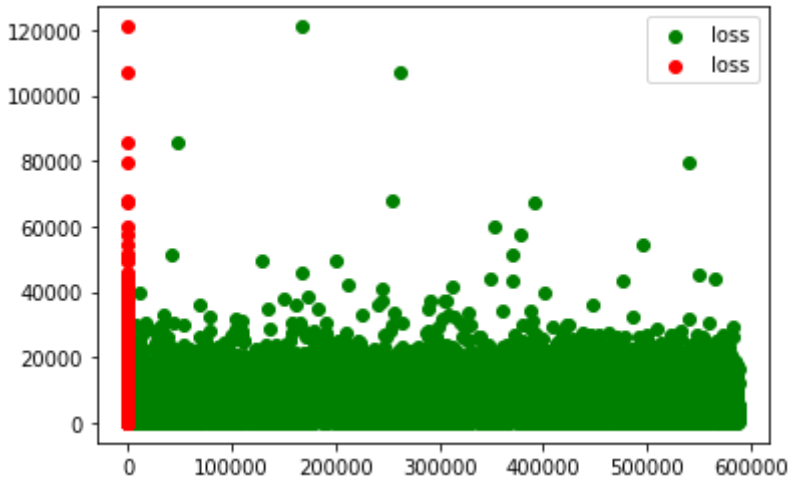
Out[15]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10
count	188318	188318	188318	188318	188318	188318	188318	188318	188318	188318
unique	2	2	2	2	2	2	2	2	2	2
top	A	A	A	A	A	A	A	A	A	A
freq	141550	106721	177993	128395	123737	131693	183744	177274	113122	160213

4 rows × 116 columns

Data Visuvalization

In [16]:
```
plt.scatter(x = 'id',y = 'loss',data = df,color = 'g')
plt.scatter(x = 'cont1',y = 'loss',data = df,color = 'r')

#Describe all the elements of a graph
plt.legend()

#show the graph
plt.show()
```



In [17]:
```
#Filling Out Numerical Columns from the Dataset

df_num = df.select_dtypes(include = [np.number])
print(df_num)
# Print heading names of columns contain only numbers
df_num.columns
```

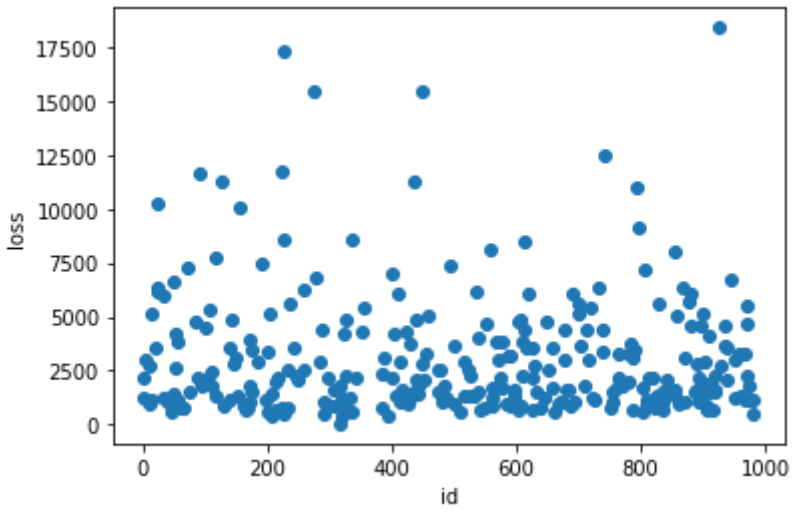
```
            id     cont1     cont2     cont3     cont4     cont5
cont6  \
0            1  0.726300  0.245921  0.187583  0.789639  0.310061
0.718367
1            2  0.330514  0.737068  0.592681  0.614134  0.885834
0.438917
2            5  0.261841  0.358319  0.484196  0.236924  0.397069
0.289648
3           10  0.321594  0.555782  0.527991  0.373816  0.422268
0.440945
4           11  0.273204  0.159990  0.527991  0.473202  0.704268
0.178193
...        ...       ...       ...       ...       ...       ...
...
188313  587620  0.347403  0.785784  0.613660  0.473202  0.939556
0.242437
188314  587624  0.507661  0.555782  0.549770  0.802892  0.704268
0.334270
188315  587630  0.484469  0.785784  0.792378  0.189137  0.482436
0.345883
188316  587632  0.438385  0.422197  0.298977  0.383428  0.340543
0.704364
188317  587633  0.907272  0.620805  0.440642  0.821574  0.281143
0.844563

           cont7    cont8    cont9   cont10    cont11    cont12
cont13  \
0       0.335060  0.30260  0.67135  0.83510  0.569745  0.594646  0.
822493
1       0.436585  0.60087  0.35127  0.43919  0.338312  0.366307  0.
611431
2       0.315545  0.27320  0.26076  0.32446  0.381398  0.373424  0.
195709
3       0.391128  0.31796  0.32128  0.44467  0.327915  0.321570  0.
605077
```

Out[17]:

```
Index(['id', 'cont1', 'cont2', 'cont3', 'cont4', 'cont5', 'cont6',
'cont7',
       'cont8', 'cont9', 'cont10', 'cont11', 'cont12', 'cont13', 'c
ont14',
       'loss'],
      dtype='object')
```

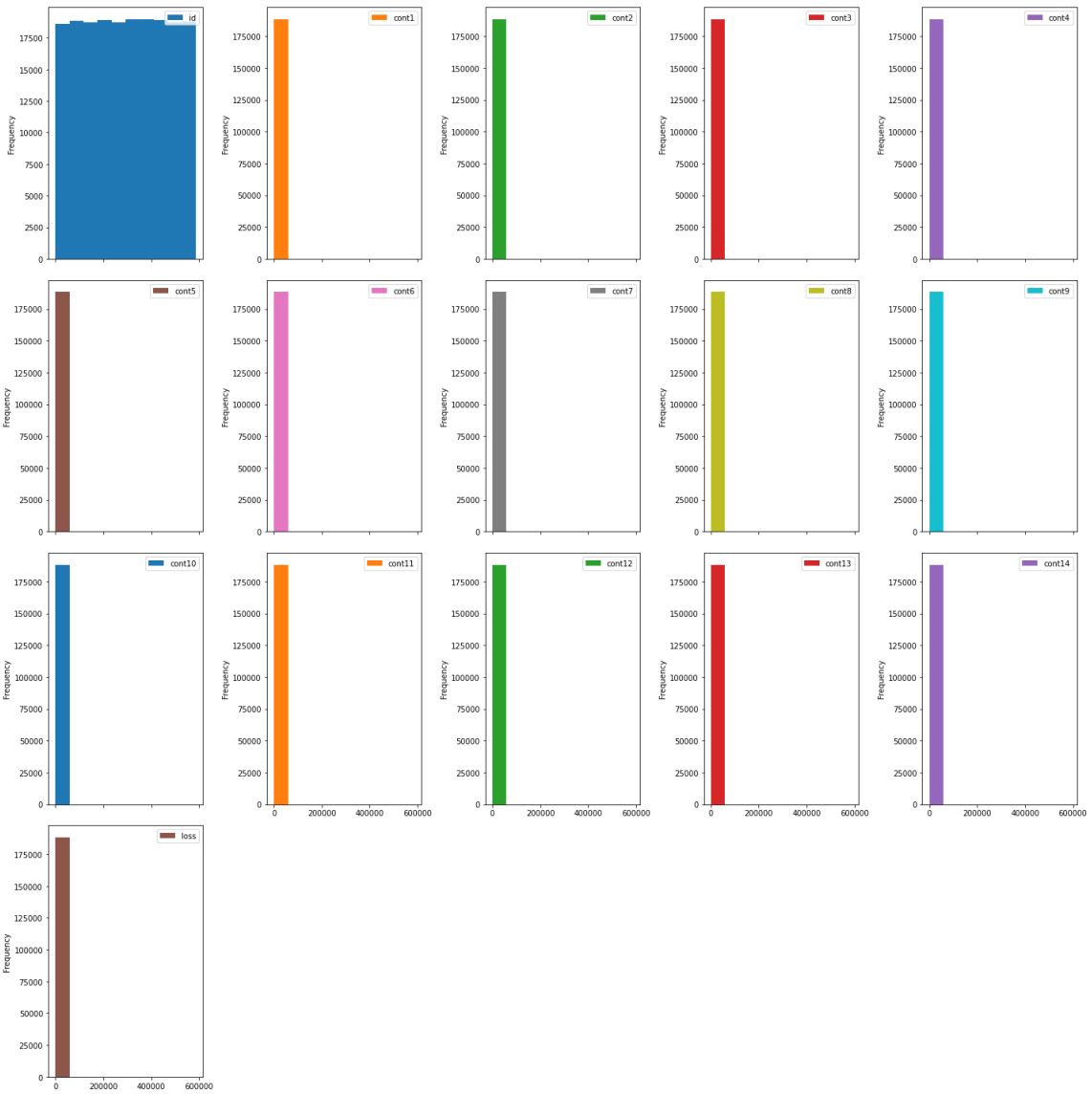
In [18]:

```
for i in df.columns[:-1]:
    plt.xlabel(i)
    plt.ylabel("loss")
    plt.scatter(df[i][:300],df["loss"][:300])
    plt.show()
```

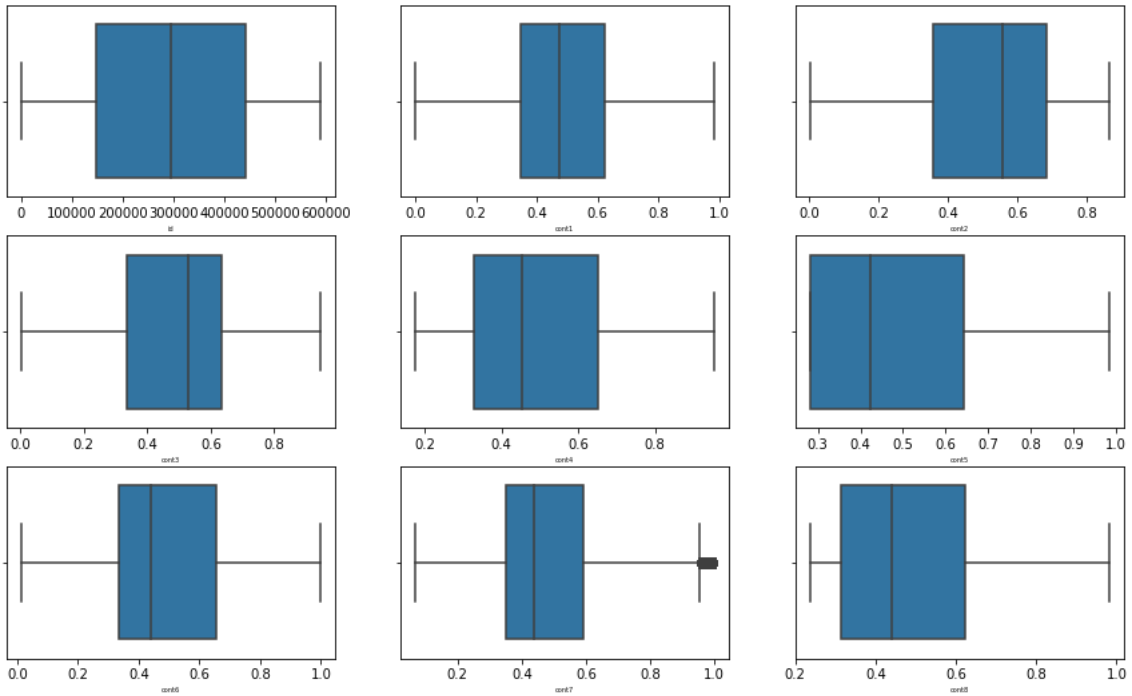


In [19]:

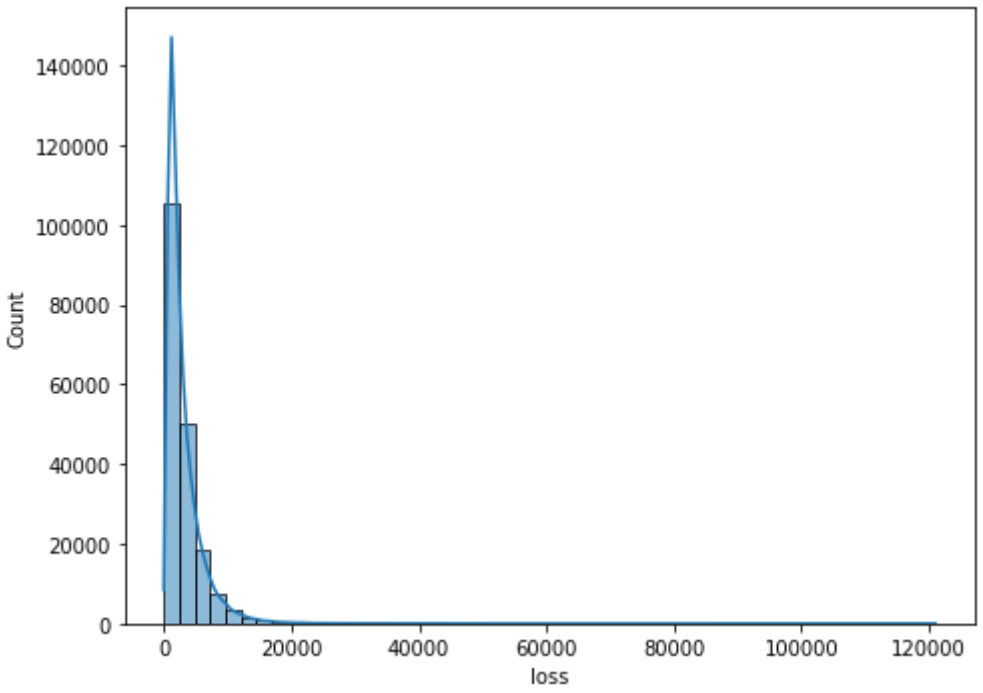
```
# plot multiple histogram for all the numerical column in my data
df.plot.hist(subplots = True , layout = (4,5),figsize=(20,20))
# rearrange the plot:
plt.tight_layout()
plt.show()
```



In [37]:
```
#To identify the Outliers in Numerical Columns:
#Subplots()
fig, ax = plt.subplots(3,3,figsize =(15,9))


for variable,subplot in zip(df_num.columns,ax.flatten()):
    z = sns.boxplot(x = df_num[variable], orient = 'h',whis = 1.5

    z.set_xlabel(variable,fontsize = 5)
```



In [20]:
```
y = df.iloc[:, -1]
plt.figure(figsize=(7, 5))
sns.histplot(data=y, kde=True, bins=50)
plt.tight_layout()
```



In [21]:
```
# Label Encoding

```

```
# The label Encoding consider a level in a Categorical variable l

from sklearn.preprocessing import LabelEncoder
# Create an Instance
labelencoder = LabelEncoder()

#Fit the Encoder
df['Encoded_performance_of_cat1'] = labelencoder.fit_transform(df

# Display the data
df
```

Out[21]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont7	cont8	
0	1	A	B	A	B	A	A	A	A	B	...	0.335060	0.30260	(
1	2	A	B	A	A	A	A	A	A	B	...	0.436585	0.60087	(
2	5	A	B	A	A	B	A	A	A	B	...	0.315545	0.27320	(
3	10	B	B	A	B	A	A	A	A	B	...	0.391128	0.31796	(
4	11	A	B	A	B	A	A	A	A	B	...	0.247408	0.24564	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
188313	587620	A	B	A	A	A	A	A	A	B	...	0.289949	0.24564	(
188314	587624	A	A	A	A	A	B	A	A	A	...	0.382000	0.63475	(
188315	587630	A	B	A	A	A	A	A	B	B	...	0.370534	0.24564	(
188316	587632	A	B	A	A	A	A	A	A	B	...	0.562866	0.34987	(
188317	587633	B	A	A	B	A	A	A	A	A	...	0.533048	0.97123	(

188318 rows × 133 columns

In [22]:
```
print(df['cat1'].value_counts())
print('\n')
df['Encoded_performance_of_cat1'].value_counts()
```

```
A    141550
B     46768
Name: cat1, dtype: int64
```

Out[22]:
```
0    141550
1     46768
Name: Encoded_performance_of_cat1, dtype: int64
```

In [23]:
```
x = np.array(df["cont1"]).reshape(-1,1)
y = np.array(df["loss"]).reshape(-1,1)
```

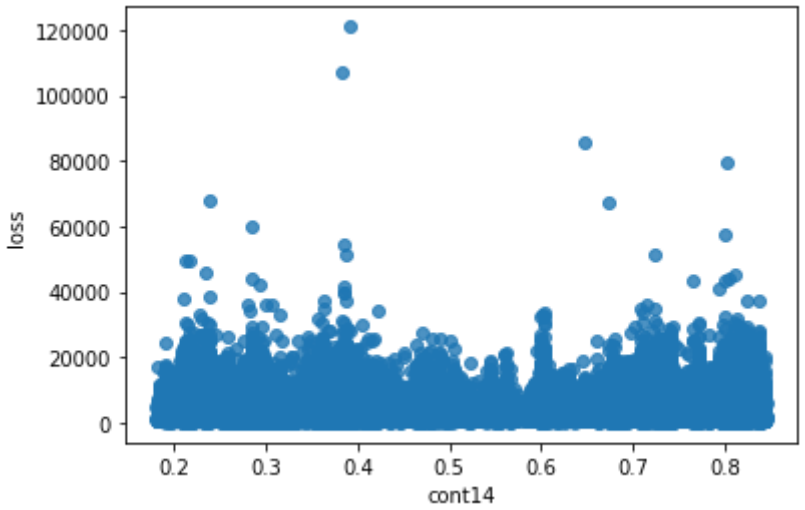
In [24]:
```
from sklearn.linear_model import LinearRegression
linear = LinearRegression()
linear.fit(x,y)
linear.predict([[12]])
```

Out[24]:
```
array([[1214.35169353]])
```

In [25]:
```
## multiple Regression
x = df.drop("loss",axis = 1)
y = df["loss"]
```

In [68]:
```
sns.regplot(x="cont14",y="loss",data = df)
```

Out[68]: <AxesSubplot:xlabel='cont14', ylabel='loss'>



In [27]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df_num.drop("loss
```

In [71]:
```
xtrain.shape
```

Out[71]: (141238, 15)

In [29]:
```
ytrain.shape
```

Out[29]: (141238,)

In [30]:
```
xtest.shape
```

Out[30]: (47080, 15)

In [31]:
```
ytest.shape
```

Out[31]: (47080,)

In [53]:
```
ytest.isnull().sum()
```

Out[53]: 0

In [32]:
```
model = LinearRegression()
```

In [33]:
```
model.fit(xtrain,ytrain)
```

Out[33]: LinearRegression()

In [34]:
```
y_pred = model.predict(xtest)
```

In [35]:
```
ytest.head(10)
```

Out[35]:

```
19407      61089
59649     186897
157826    491890
74298     232234
168874    526714
100911    315340
```

In [36]:
```
1 xtest.head(10)
```

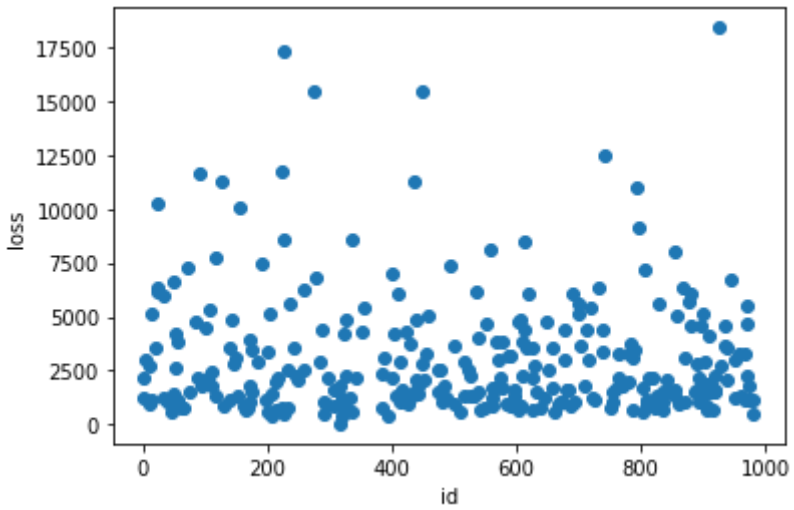
Out[36]:

	id	cont1	cont2	cont3	cont4	cont5	cont6	cont7	con
19407	61089	0.513457	0.681761	0.549770	0.594598	0.811271	0.416181	0.402349	0.629
59649	186897	0.703859	0.488789	0.263570	0.623770	0.783230	0.874303	0.687567	0.645
157826	491890	0.355333	0.681761	0.692825	0.284048	0.281143	0.526202	0.868961	0.435
74298	232234	0.329232	0.358319	0.484196	0.373816	0.422268	0.372125	0.388545	0.317
168874	526714	0.546670	0.681761	0.728827	0.373816	0.422268	0.364464	0.401162	0.268
100911	315340	0.936584	0.245921	0.120071	0.661283	0.281143	0.903531	0.553910	0.829
148414	462856	0.292010	0.737068	0.634224	0.623770	0.281143	0.314937	0.413673	0.672
188085	586944	0.340859	0.785784	0.506105	0.267727	0.960658	0.369930	0.405795	0.476
24040	75564	0.475784	0.555782	0.462347	0.534409	0.889301	0.373500	0.347485	0.360
42362	132985	0.218540	0.488789	0.397983	0.524230	0.281143	0.342167	0.291170	0.349

In [37]:
```
1 for i in y_pred[:10]:
2     print(i)
```

```
61089.000000001055
186897.00000000047
491889.999999999
232234.00000000032
526713.9999999988
315339.99999999994
462855.9999999992
586943.9999999986
75564.00000000102
132985.00000000073
```

In [64]:
```
for i in df.columns[:-1]:
    plt.xlabel(i)
    plt.ylabel("loss")
    plt.scatter(df[i][:300],df["loss"][:300])
    plt.show()
```

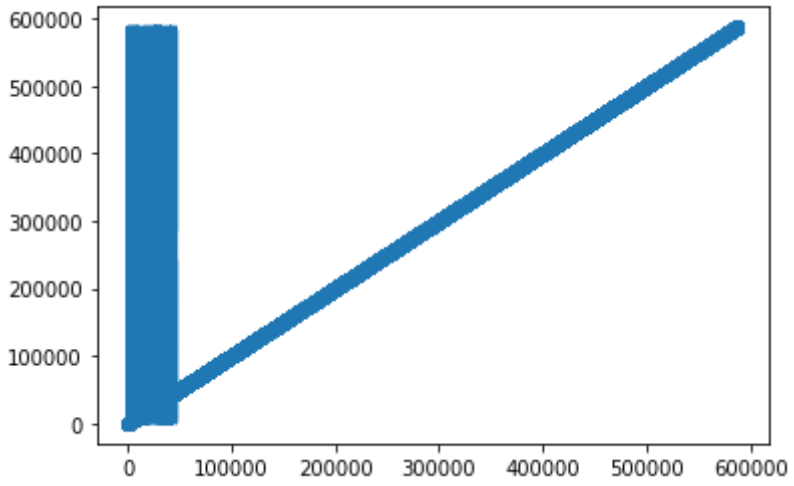


In [38]:
```
model.score(xtrain,ytrain)
```

Out[38]: 1.0

In [94]:
```
plt.scatter(xtest["id"],ytest)
plt.plot(y_pred)
```

Out[94]: [<matplotlib.lines.Line2D at 0x7f1d3773a5c0>]



In [87]:
```
### Polynomial Regression
from sklearn.preprocessing import PolynomialFeatures
lin = LinearRegression()
poly = PolynomialFeatures(degree = 2)
x = np.array(df["id"]).reshape(-1,1)
y = np.array(df["loss"]).reshape(-1,1)
x_train = poly.fit_transform(x)
```

In [88]:
```
x_train
```

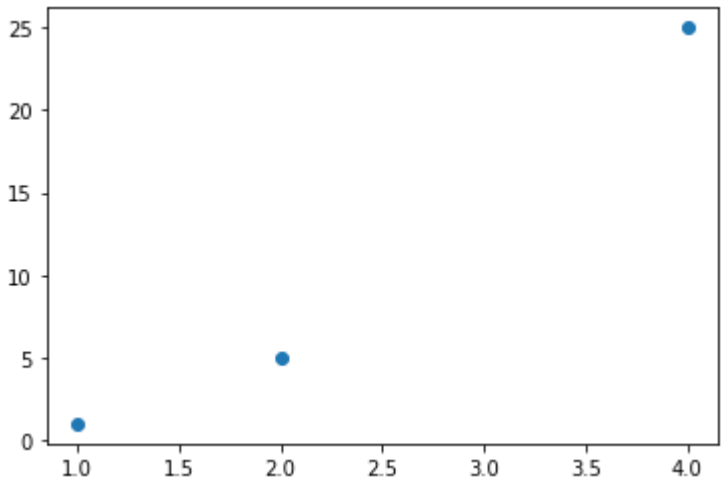
Out[88]:
```
array([[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
       [1.00000000e+00, 2.00000000e+00, 4.00000000e+00],
       [1.00000000e+00, 5.00000000e+00, 2.50000000e+01],
       ...,
       [1.00000000e+00, 5.87630000e+05, 3.45309017e+11],
       [1.00000000e+00, 5.87632000e+05, 3.45311367e+11],
       [1.00000000e+00, 5.87633000e+05, 3.45312543e+11]])
```

In [89]:
```
x_train.shape
```

Out[89]:
```
(188318, 3)
```

In [90]:
```
plt.scatter(x_train[1],x_train[2])
```

Out[90]:
```
<matplotlib.collections.PathCollection at 0x7f1d374656a0>
```



In [91]:
```
from sklearn.linear_model import LinearRegression
lin = LinearRegression()
print(poly.fit(x_train,y))
print(lin.fit(x_train,y))
lin.intercept
```

```
PolynomialFeatures()
LinearRegression()
```

Out[91]:
```
array([3031.94493943])
```

In [40]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_erro
```

In [41]:
```
mean_absolute_error(ytest,y_pred)
```

Out[41]:
```
6.963696855254863e-10
```

In [42]:
```
mean_squared_error(ytest,y_pred)
```

Out[42]:
```
6.436651276460588e-19
```

In [43]:
```
model.score(xtrain,ytrain)
```

Out[43]:
```
1.0
```

In [ ]:
```
1
```