In [1]:
```
!pip install pandas
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
%matplotlib inline
import math
import numpy as np
from sklearn.decomposition import PCA
```

```
Requirement already satisfied: pandas in c:\anaconda\lib\site-packages (1.0.1)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\anaconda\lib\site-p
ackages (from pandas) (2.8.1)
Requirement already satisfied: numpy>=1.13.3 in c:\anaconda\lib\site-packages
(from pandas) (1.18.1)
Requirement already satisfied: pytz>=2017.2 in c:\anaconda\lib\site-packages (f
rom pandas) (2019.3)
Requirement already satisfied: six>=1.5 in c:\anaconda\lib\site-packages (from
python-dateutil>=2.6.1->pandas) (1.14.0)
```

In [47]:
```
df=pd.read_csv('C:\\Users\Admin\\Downloads\\dataset_00_with_header (1).csv')
```

In [48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 305 entries, x001 to y
dtypes: float64(41), int64(264)
memory usage: 232.7 MB
```

In [49]:
```
df.describe()
```

Out[49]:

	x001	x002	x003	x004	x005	x006	
count	1.000000e+05	78568.000000	78568.000000	78576.000000	93890.000000	100000.000000	1000
mean	1.218244e+06	125.711727	25.541238	65.393212	178.238545	0.314040	
std	2.728977e+05	115.785117	49.028751	63.592317	124.520628	0.464135	
min	5.170000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	9.743635e+05	32.000000	3.000000	19.000000	87.000000	0.000000	
50%	1.235926e+06	100.000000	8.000000	48.000000	150.000000	0.000000	
75%	1.445326e+06	180.000000	24.000000	92.000000	246.000000	1.000000	
max	1.677197e+06	718.000000	704.000000	704.000000	827.000000	1.000000	

8 rows × 305 columns

In [50]:
```
df.shape
```
Out[50]:
```
(100000, 305)
```

In [51]:
```
df.head()
```
Out[51]:

	x001	x002	x003	x004	x005	x006	x007	x008	x009	x010	...	x296	x297	x298	x299
0	1540332	NaN	NaN	NaN	8.0	1	0	1	0	0	...	0	NaN	0	0
1	823066	4.0	3.0	3.0	4.0	0	2	2	0	0	...	5206	0.9339	1	1
2	1089795	NaN	NaN	NaN	96.0	1	0	0	0	1	...	0	NaN	0	0
3	1147758	63.0	14.0	38.0	258.0	0	0	0	1	2	...	0	NaN	1	1
4	1229670	34.0	25.0	29.0	34.0	1	0	0	0	3	...	0	NaN	0	0

5 rows × 305 columns

In [52]:
```
dict(df.dtypes)
```
```
 'x012': dtype('int64'),
 'x013': dtype('int64'),
 'x014': dtype('int64'),
 'x015': dtype('int64'),
 'x016': dtype('int64'),
 'x017': dtype('int64'),
 'x018': dtype('int64'),
 'x019': dtype('int64'),
 'x020': dtype('int64'),
 'x021': dtype('int64'),
 'x022': dtype('int64'),
 'x023': dtype('int64'),
 'x024': dtype('int64'),
 'x025': dtype('int64'),
 'x026': dtype('int64'),
 'x027': dtype('int64'),
 'x028': dtype('int64'),
 'x029': dtype('int64'),
 'x030': dtype('int64'),
 'x031': dtype('int64'),
```

In [53]:
```

feature_cols = [col for col in df.columns if col!="y"]
```

In [54]:
```
print("No. of columns having the null values: ",df.isnull().any().sum())
```
```
No. of columns having the null values:  41
```

In [55]:
```
# Assuming that the columns which are having less than 50 unique values are categ
cat_cols = []
for col in feature_cols:
    if df[col].nunique()<200:
        cat_cols.append(col)
```

In [56]:
```
len(cat_cols)
```

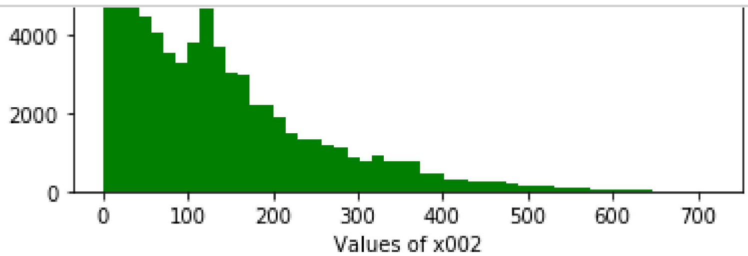
Out[56]: 248

In [57]:
```
num_cols = list(set(feature_cols).difference(set(cat_cols)))
```

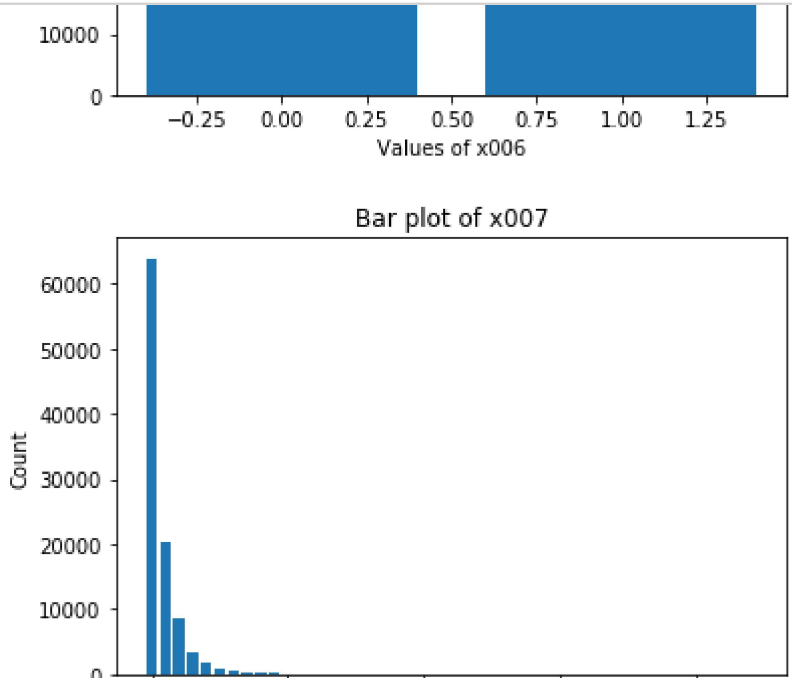
In [58]:
```
null_dict = {col: null_count for col, null_count in dict(df.isnull().sum()).items
```

In [59]:
```
# Hitogram for distribution of values in the columns having null values

for col in null_dict.keys():
    plt.hist(df[col],bins = 50,color = "green")
    plt.xlabel(f"Values of {col}")
    plt.ylabel("Count")
    plt.title(f"Histogram of {col}")
    plt.show()
```



In [61]:
```
# Bar plot for categorical columns
for col in cat_cols:
    plt.bar(dict(df[col].value_counts()).keys(),dict(df[col].value_counts()).valu
    plt.xlabel(f"Values of {col}")
    plt.ylabel("Count")
    plt.title(f"Bar plot of {col}")
    plt.show()
```



In [62]:
```
# null columns which are categorical
null_cats = list(set(null_dict.keys()).intersection(cat_cols))
```

In [63]:
```
# Null cols which are numerical
null_nums = list(set(null_dict.keys()).difference(null_cats))
```

In [64]:
```
# Imputing the null values in the categorical column with mode
for col in null_cats:
    df[col] = df[col].fillna(df[col].mode()[0])
```

In [65]:
```
# Imputing the null values in the numerical columns with the column median

for col in null_nums:
    df[col] = df[col].fillna(df[col].median())
```

In [66]:
```
print("No. of columns having the null values after imputation :",df.isnull().any(
```

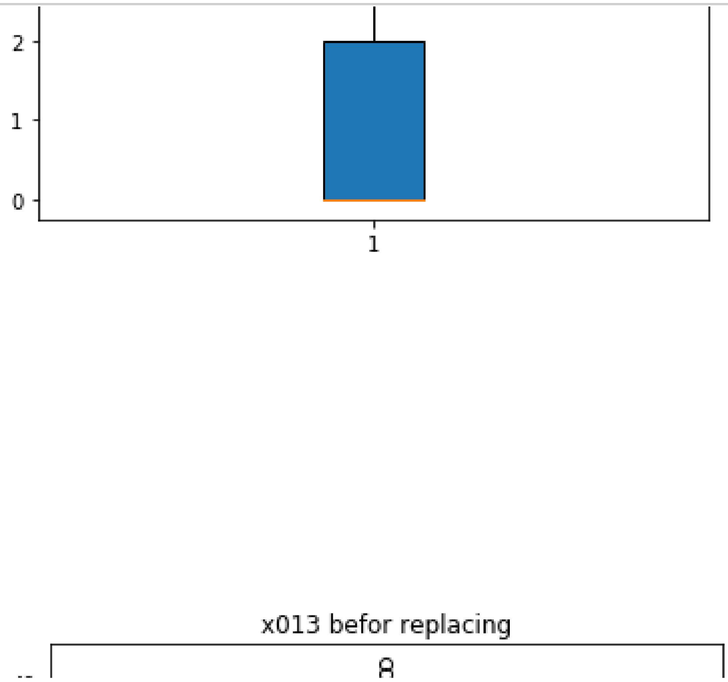
```
No. of columns having the null values after imputation : 0
```

In [67]:
```
#co="x001"
```

In [68]:
```
for col in num_cols:
    q1,q3=np.percentile(df[col],[25,75])
    iqr = q3-q1
    lower=q1-(1.5*iqr)
    upper=q3 + 1.5*iqr
    #print(q1,q3,lower,upper)
    #plt.boxplot(df[col],patch_artist=True)
    #plt.title(f"{col} befor replacing")
    #plt.show()
    #sns.boxplot(data=df,x=df[col])
    df.loc[(df[col]<lower) | (df[col]>upper), col] = df[col].median()
    #sns.boxplot(data=df,x=df[col])
    #plt.boxplot(df[col],patch_artist=True)
    #plt.title(f"after {col} replacing outlier")
    #plt.show()

    #np.where((df[col]<lower) | (df[col]>upper), df[col].median(), df[col])
#df.loc[(df[col]<lower) | (df[col]>upper), col] = df[col].median()
#dict((df[col]<lower) | (df[col]>upper))
#print(df[col].median())
#dict((df[col]<lower) | (df[col]>upper))
#df[col].replace(df[(df[col]<lower) | (df[col]>upper)][col],df[col].median())
#df.loc[df['col'] > 1990, 'First Season'] = 1
#df[col] = np.where((df[col]<lower) | (df[col]>upper),df[col].median(),df[col])
```

In [69]:
```python
for col in cat_cols:
    q1,q3=np.percentile(df[col],[25,75])
    iqr = q3-q1
    lower=q1-(1.5*iqr)
    upper=q3 + 1.5*iqr
    #print(q1,q3,lower,upper)
    plt.boxplot(df[col],patch_artist=True)
    plt.title(f"{col} befor replacing")
    plt.show()
    #sns.boxplot(data=df,x=df[col])
    df.loc[(df[col]<lower) | (df[col]>upper), col] = df[col].mode()[0]
    #sns.boxplot(data=df,x=df[col])
    plt.boxplot(df[col],patch_artist=True)
    plt.title(f"after {col} replacing outlier")
    plt.show()
```



In [71]:
```python
print("No. of columns having the null values after imputation :",df.isnull().any(
```

```
No. of columns having the null values after imputation : 0
```

In [72]:
```
df
```

Out[72]:

	x001	x002	x003	x004	x005	x006	x007	x008	x009	x010	...	x296	x297	x29
0	1540332.0	100.0	8.0	48.0	8.0	1	0	1	0	0	...	0.0	0.851	
1	823066.0	4.0	3.0	3.0	4.0	0	2	2	0	0	...	5206.0	0.851	
2	1089795.0	100.0	8.0	48.0	96.0	1	0	0	0	1	...	0.0	0.851	
3	1147758.0	63.0	14.0	38.0	258.0	0	0	0	1	2	...	0.0	0.851	
4	1229670.0	34.0	25.0	29.0	34.0	1	0	0	0	0	...	0.0	0.851	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99995	1573467.0	200.0	3.0	48.0	200.0	1	0	3	0	0	...	30960.0	0.851	
99996	1653422.0	292.0	8.0	48.0	292.0	1	1	1	1	2	...	0.0	0.851	
99997	1284669.0	35.0	4.0	26.0	57.0	0	1	1	5	0	...	0.0	0.851	
99998	1434877.0	4.0	3.0	3.0	4.0	0	2	2	0	0	...	0.0	0.851	
99999	1596945.0	134.0	19.0	75.0	150.0	0	0	0	1	1	...	12733.0	0.851	

100000 rows × 305 columns

In [27]:
```
df.isnull().any().sum()
```

Out[27]: 0

In [74]:
```

standerd_scaler =  StandardScaler()

def scaleColumns(df, cols_to_scale):
    for col in cols_to_scale:
        df[col] = pd.DataFrame(standerd_scaler.fit_transform(pd.DataFrame(df[col]
    return df

df_scale=scaleColumns(df,feature_cols)
```

In [75]:
```
df_scale.shape
```

Out[75]: (100000, 305)

In [76]:
```
df.isnull().any().sum()
```

Out[76]: 0

In [77]:
```
pca = PCA(n_components=2)
pca.fit(df_scale)
```

Out[77]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [78]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

Out[78]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [79]:
```
columns = ['pca_%i' % i for i in range(2)]
X = pd.DataFrame(pca.transform(df_scale), columns=columns, index=df.index)
X.head()
```

Out[79]:

	pca_0	pca_1
0	-86.708429	-6.364956
1	61.305422	0.409677
2	42.255420	-3.700428
3	93.222871	2.047789
4	123.261267	-2.298389

In [80]:
```
y=df["y"]
```

In [81]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X,y, test_size=0.3, random_s
```

In [82]:
```
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def cross_val(model):
    pred = cross_val_score(model, X, y, cv=10)
    return pred.mean()

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('__________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

In [83]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train,y_train)
```

Out[83]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [84]:
```
#print the intercept
print(lin_reg.intercept_)
```

```
619.1983608820225
```

In [85]:
```
coeff_df = pd.DataFrame(lin_reg.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

Out[85]:

	Coefficient
pca_0	-0.999645
pca_1	-0.010057

In [86]:
```
lin_reg.coef_
```

Out[86]:
```
array([-0.99964483, -0.01005702])
```

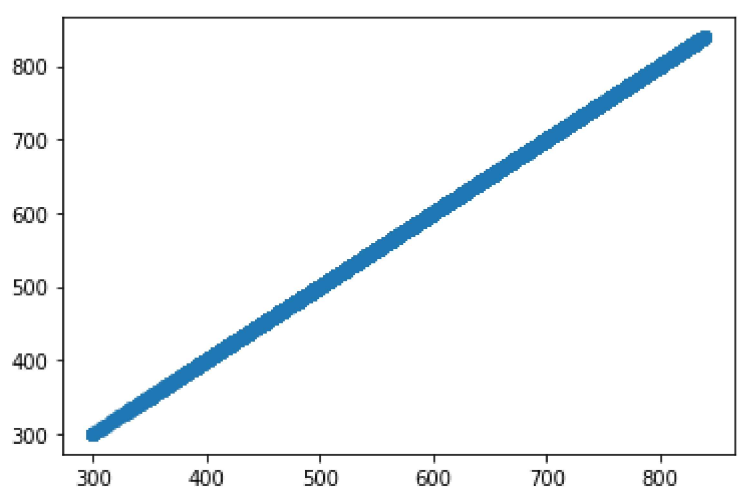
In [87]:
```
X.columns
```

Out[87]:
```
Index(['pca_0', 'pca_1'], dtype='object')
```

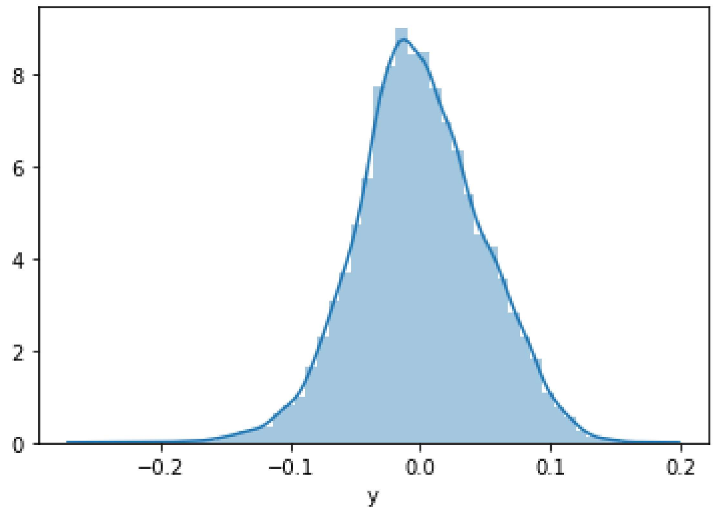
In [88]:
```
pred = lin_reg.predict(X_test)
```

In [89]:
```
plt.scatter(y_test, pred)
```

Out[89]:
```
<matplotlib.collections.PathCollection at 0x1dc74fc6048>
```



In [90]:
```
sns.distplot((y_test - pred), bins=50);
```



In [91]:
```
test_pred = lin_reg.predict(X_test)
train_pred = lin_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

```
Test set evaluation:
_____________________________________
MAE: 0.038159398435530414
MSE: 0.0023320421806113825
RMSE: 0.04829122260423091
R2 Square 0.999999834017107
__________________________________
Train set evaluation:
_____________________________________
MAE: 0.03805335179532369
MSE: 0.0023040949127598022
RMSE: 0.048000988664399426
R2 Square 0.999999835723747
__________________________________
```

In [92]:
```
results_df = pd.DataFrame(data=[["Linear Regression", *evaluate(y_test, test_pred
                          columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', "C
results_df
```

Out[92]:

	Model	MAE	MSE	RMSE	R2 Square	Cross Validation
0	Linear Regression	0.038159	0.002332	0.048291	1.0	1.0