In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
import pandas as pd
train_file =r"C:\Users\hii\Desktop\data science\train.csv"
train_df=pd.read_csv(train_file)
test_file=r"C:\Users\hii\Desktop\data science\test.csv"
test_df=pd.read_csv(test_file)
```

In [ ]:
```
train_df
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
test_df.head()
```

In [ ]:
```
train_df.columns.value_counts
```

In [ ]:
```
train_df.describe().T
```

In [ ]:
```
train_df.isnull().sum()
```

In [ ]:
```
train_df.dtypes
```

In [ ]:
```
train_df.info()
```

In [ ]:
```
int_col = len(train_df.select_dtypes(include=['int64']).columns)
flt_col = len(train_df.select_dtypes(include=['float64']).columns)
obj_col = len(train_df.select_dtypes(include=['object']).columns)
```

In [ ]:
```
print("The No of integer columns are:", int_col)
print("The No of object columns are:", obj_col)
print("The No of float columns are:", flt_col)
```

In [ ]:
```
int_col = len(test_df.select_dtypes(include=['int64']).columns)
obj_col = len(test_df.select_dtypes(include=['float64']).columns)
flt_col = len(test_df.select_dtypes(include=['object']).columns)
```

In [ ]:
```
print("The No of integer Features are:", int_col)
print("The No of object Features are:", obj_col)
print("The No of float Features are:", flt_col)
```

In [ ]:
```
train_df.drop("ID",inplace=True,axis=1)
test_df.drop("ID",inplace=True,axis=1)
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
train_df.var()[train_df.var() == 0].index.values
```

In [ ]:
```
threshold = 0.0
train_df.drop(train_df.var()[train_df.var() == threshold].index.values, axis=1,inp
```

In [ ]:
```
train_df.shape
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
test_df.drop(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290','X293',
```

In [ ]:
```
test_df.shape
```

In [ ]:
```
train_df_num=train_df.select_dtypes(include=["int64"])
```

In [ ]:
```
train_df_cat=train_df.select_dtypes(include=["object"])
```

In [ ]:
```
train_df_cat
```

In [ ]:
```
train_df_num
```

In [ ]:
```
sns.boxplot(train_df.y)
```

In [ ]:
```
def outlier_detection(train_df_column):
    sorted(train_df_column)
    Q1, Q3=np.percentile(train_df_column,[25,75])
    IQR = Q3-Q1
    lower_range = Q1-(1.5*IQR)
    upper_range = Q3+(1.5*IQR)

    return lower_range, upper_range
```

In [ ]:
```
l,u = outlier_detection(train_df.y)
```

In [ ]:
```
print("Lower acceptable range value is ", l)
print("Upper acceptable range value is ", u)
```

In [ ]:
```
train_df.drop(train_df[(train_df.y<l)|(train_df.y>u)].index, inplace=True)
```

In [ ]:
```
split_train
sns.boxplot(train_df.y)
```

In [ ]:
```
train_df.describe(include='object')
```

In [ ]:
```
label_columns= train_df.describe(include=["object"]).columns.values
label_columns
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [ ]:
```
obj_cols = train_df.describe(include='object').T.index.values
for i in obj_cols:
  le.fit(train_df[i].append(test_df[i]).values) # appending test values for same l
  train_df[i] = le.transform(train_df[i])
  test_df[i] = le.transform(test_df[i])
```

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

In [ ]:
```
x=train_df.drop("y",axis=1)
```

In [ ]:
```
y=train_df[["y"]]
```

In [ ]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=4,test_size=.2)
```

In [ ]:
```
print("The x_train shape is:",x_train.shape)
print("The x_test shape is:",x_test.shape)
print("The y_train shape is:",y_train.shape)
print("The y_test shape is:",y_test.shape)
```

In [ ]:
```
x_test.head()
```

In [ ]:
```
x_train.head()
```

In [ ]:
```
from sklearn.decomposition import PCA
```

In [ ]:
```
pca=PCA(0.98,svd_solver="full")
```

In [ ]:
```
pca.fit(x)
```

In [ ]:
```
pca.n_components_
```

In [ ]:
```
pca.explained_variance_ratio_
```

In [ ]:
```
pca_x_train= pd.DataFrame(pca.transform(x_train))
pca_x_test=pd.DataFrame(pca.transform(x_test))
test_df_data = pd.DataFrame(pca.transform(test_df))
```

In [ ]:
```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error,r2_score
```

In [ ]:
```
model = xgb.XGBRegressor(objective='reg:linear',learning_rate=0.1)
model.fit(pca_x_train,y_train)
xgpred=model.predict(pca_x_test)
```

In [ ]:
```
print("For XGBOOST ......")
print("The RMSE value is",np.sqrt(mean_squared_error(y_test,xgpred)))
print("r_square value is", r2_score(y_test,xgpred))
```

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

In [ ]:
```
lr=LinearRegression()
lr.fit(pca_x_train,y_train)
pred=lr.predict(pca_x_test)
```

In [ ]:
```
print("for Linear Regression Model......")
print("The RMSE for Linear Model",np.sqrt(mean_squared_error(y_test,pred)))
print("r_square value is", r2_score(y_test,pred))
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rfr_model = RandomForestRegressor()
rfr_model.fit(pca_x_train,y_train)
print("For Random Forest Regressor -")
y_pred_rfr = rfr_model.predict(pca_x_test)
```

In [ ]:
```
print("for Random Forest Regressor ........")
print("R2 Score =",r2_score(y_test,y_pred_rfr))
print("Root Mean Squared Error for Random Forest Regressor =",np.sqrt(mean_squared_
```

In [ ]:
```
from sklearn.svm import LinearSVR
svr_model = LinearSVR()
svr_model.fit(pca_x_train,y_train)

y_pred_svr = svr_model.predict(pca_x_test)
```

In [ ]:
```
print("For Linear Support Vector Regression -")
print("R2 Score =",r2_score(y_test,y_pred_svr))
print("Root Mean Squared Error for SVC =",np.sqrt(mean_squared_error(y_test,y_pred_
```

In [ ]:
```
test_df_data.head()
```

In [ ]:
```

```

In [ ]:
```

```