Step For Doing ML Algorithm

1. Import All Libraries

    ```
    pd.set_option('display.max_columns', None)
    pd.set_option('display.max_rows', None)
    ```

    For Eliminate All Error:

    ```
    import warnings
    warnings.simplefilter('ignore')
    ```

2. Import Data Set (pd.read_csv() , pd.read_excel() etc..)
    - If you import from sklearn then do following :

    ```
    from sklearn.datasets import load_iris
    import pandas as pd
    data = load_iris()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df.head()
    ```

3. Take some insight like
    a. Data.shape , Data.head , Data.tail , Data.info , Data.describe
4. Analysis EDA
    a. See distribution of plot (Regression) is it is Gaussian or not or which skew is it for target variable

    ```
    #  Distribution of the target variable (Regression)
    sns.distplot(data.target, bins = 25)
    ```

    b. In (classification) we do for count plot to see our target variable imbalanced or balanced

    ```
    sns.countplot(data['target'])
    ```

5. Find out all numerical predictor

    ```
    numeric_features = data.select_dtypes(include=[np.number])
    numeric_features.dtypes
    ```

6. Then 1st find correlation between all numerical predictor

    ```
    # Correlation between all

    corr =numeric_features.corr()

    plt.figure(figsize=(12,10))

    sns.heatmap(corr, vmax=.8, square=True,annot=True)

     #correlation between features and target variable?

    corr_matrix = abs(df.corr())

    print(corr_matrix["target_variable"].sort_values(ascending=False))
    ```

7. Then find all categorical feature

   ```
   cat_features = train.select_dtypes(include=[np.object])
   cat_features.dtypes
   ```

8. See distribution among all the feature
   a. Like sns.countplot(train.Age)
   b. And use pair plot and other type of plot
9. Treat missing value(mean ,median , mode , drop,0)
10. Then treat them apply onehot , dummy, label encode etc
11. Outlier Detection (By boxplot)
12. In this step we treat all null and categorical value
13. Then do some feature engineering / feature selection
14. Store target variable in y and all variable in X
15. Then if in data imbalanced we 1st balanced data by using SMOTE

#Step 1: Here I use Oversampling Using Smote

```
from imblearn.over_sampling import SMOTE

x_resample, y_resample  = SMOTE().fit_sample(x, y.values.ravel())

# lets print the shape of x and y after resampling it

print(x_resample.shape)

print(y_resample.shape)
```

#Step 2:

```
# lets also check the value counts of our target variable4

print("Before Resampling :")

print(y.value_counts())

print("After Resampling :")

y_resample = pd.DataFrame(y_resample)

print(y_resample[0].value_counts())
```

16. Then scale the data by using standard scalar :

```
# lets import the standard scaler library from sklearn to do that

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)
```

```
x_valid = sc.transform(x_valid)
```

17.Then we split train_test_split
18.Model Build (RF,DT,LR,LoR,XGB)
19. Apply K-Flod cross validation (if train data is few)
20.Apply Hyper parameter tuning
21.Find all accuracy Both(Regrssion,Classification)

a. Cross Validation :

Here I Use StratifiedKFold :

```
kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=45)

pred_test_full =0

cv_score =[]

i=1

for train_index,test_index in kf.split(X,y):

    print('{} of KFold {}'.format(i,kf.n_splits))

    #Split X_train,X_test,y_train,y_test

    xtr,xvl = X.loc[train_index],X.loc[test_index]

    ytr,yvl = y.loc[train_index],y.loc[test_index]

    #model

     model = #add any model

    model.fit(xtr,ytr)

    #performance Calculation

    score = roc_auc_score(yvl,model.predict(xvl))

    print('ROC AUC score:',score)

    cv_score.append(score)

    #predict test data

    pred_test = model.predict_proba(x_test)[:,1]

    pred_test_full +=pred_test

     i+=1
```

b. Hyper Parameter Tuning :

## initialize all parameter

```
params={
"max_depth" : [50,100,150,200],
"min_samples_split" : [1,2,3,4,5,6,7,8,9],
"min_samples_leaf" : [1,2,3,4,5,6,7,8,9],
"criterion": ["gini", "entropy"],
"max_leaf_nodes":[1,5,10,15,20]}
```

## Apply desire model for tuning

```
classifier= DecisionTreeClassifier()
```

## Randomize search Cv is start

```
random_search = RandomizedSearchCV (classifier,param_distributions=params,n_iter
=5,scoring='roc_auc',n_jobs=-1,cv=5,verbose=3)
random_search.fit(X,y)
```

## Calculate The Score and best params

```
print('Best roc_auc: {:.4}, with best C: {}'.format(random_search.best_score_,
random_search.best_params_))
```

c. Calculate Performance Matric :

For Regression r2_score,MAE,RMSE

```
from sklearn.metrics import  r2_score,mean_squared_error,mean_absolute_error
from math import sqrt

from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()

rfr.fit(X_train,y_train)

#Predicting the Test set results
y_pred_rfr = rfr.predict(X_train)
score = r2_score(y_train,y_pred_rfr)
print("Score of Training:",100*score)
print("RMSE :" , np.sqrt(mean_squared_error(y_train,y_pred_rfr)))

y_test_pred_rfr = rfr.predict(X_test)
#r2 Score
score = r2_score(y_test,y_test_pred_rfr)
print("Score of Testing:",100*score)
#RMSE
print("RMSE : " , np.sqrt(mean_squared_error(y_test,y_test_pred_rfr)))
#MAE
print("Mean Absolute Error",mean_absolute_error(y_test,y_test_pred_rfr))
```

For Classification (confusion matrix, accuracy, precision, recall)

```
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

print("Training Accuracy :", lr.score(X_train, y_train))
print("Testing Accuracy :", lr.score(X_test, y_test))

#Confusion Matric
cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot = True, cmap = 'Wistia', fmt = '.8g')
plt.show()

#Classification Report
cr = classification_report(y_test, y_pred)
print(cr)
```

------------------- --------------------------------- ----------------------------------- ----------------------

## Using Pickle Model TO Dumb and Load Model:

```
#dumb file
import pickle
filename = 'filename.pkl’
pickle.dump(model_instance, open(filename, 'wb'))
```

```
#open file
model = open("cc_strength.pkl","rb")
model = pickle.load(model)
```

-------------------- ------------ ------------------------- --------------------- ---------------