-> check df.info()
-> find the null values of each col
-> check for duplicates and drop
-> check for random special characters

-> Handling Missing Values
    -> leave as it is

    -> fill the missing values
        -> df.fillna()
        -> Simple Imputer
            -> replaces NaN values with specified placeholder
            -> from sklearn.impute import SimpleImputer
                imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
                df['workclass'] = imputer.fit_transform(df[['workclass']]).ravel()

-> drop the missing values
    df.dropna(axis=0, how="any", thresh=None,
    subset=None, inplace=False)

-> replace the special char with `
    df = df.replace("?", np.nan)
    np.nan is na

-> hadleling categorical data
    ->replace
        -> print(df['gender'].unique())
            df['gender']=df['gender'].replace('Male',1)
            df['gender']=df['gender'].replace('Female',0)

-> Label Encoder
    -> fills the NaN values with random numeric values
    -> from sklearn.preprocessing import LabelEncoder
        lbl_encoder=LabelEncoder()
        temp_df['Fruit_Name']=lbl_encoder.fit_transform(temp_df['Fruit_Name'])

-> One hot encoding

-> Check for Outliers

-> use boxplot

-> sns.boxplot(df['age'])

-> use scatterplot

-> use zscore

-> IQR

->Q1 = df.quantile(0.25)

Q3 = df.quntile(0.75)

IQR = Q3-Q1

lwr_bound = Q1-(1.5*IQR)

upr_bound = Q3+(1.5*IQR)

print("min: ", lwr_bound, " Max: ", upr_bound)

outliers=[]

for i in df['age']:

if(i<lwr_bound or i>upr_bound):

outliers.append(i)

print("No. of outliers: ",len(outliers))

print(outliers)

-> Handling Outliers (reamaing)

-> Removing

-> quartile based flooring and capping

-> mean/median imputation

-> Data Sampling

-> lt_fifty_k=df[df['income']==0]

gt_fifty_k=df[df['income']==1]

print("<=50k: ", lt_fifty_k.shape)

print(">50k: ", gt_fifty_k.shape)

-> no_sample=lt_fifty_k.sample(n=11681)

-> concat the df

sampled_df=pd.concat([no_sample,gt_fifty_k],axis=0)

# MODEL TRAINING

-> Create X and Y data

```
X = df.drop('income',axis=1)
y = df['income']
```

-> Find the corelation of the attributes in the dataset and drop the useless columns

-> df.corr()

-> Using Mutual info - shows on which column the target value is is depended the most

-> from sklearn.feature_selection import mutual_info_classif

```
# determine the mutual information
mutual_info = mutual_info_classif(X, y)
mutual_info
```

-> mutual_info = pd.Series(mutual_info)

```
mutual_info.index = X.columns
mutual_info.sort_values(ascending=False)
```

| | |
|---|---|
| relationship | 0.115342 |
| marital-status | 0.109612 |
| capital-gain | 0.082051 |
| age | 0.065713 |
| educational-num | 0.065588 |
| education | 0.064791 |
| occupation | 0.053871 |
| hours-per-week | 0.042513 |
| fnlwgt | 0.038678 |
| capital-loss | 0.035040 |
| gender | 0.030031 |
| workclass | 0.018267 |
| race | 0.013916 |
| native-country | 0.008430 |

so drop native-country, race, workcalss, gender, capital loss

Train Test Split

-> from sklearn.model_selection import train_test_split
x_temp=temp_df.drop('Fruit_Price',axis=1)
y_temp=temp_df['Fruit_Price']

-> xtrain, xtest,ytrain,ytest=train_test_split(x_temp,y_temp,test_size=0.5,random_state=0,shuffle=True)

-> X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42,shuffle=True)

Data Scaling

-> Standard Scalar

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

X_train_std=scaler.fit_transform(X_train)
X_test_std=scaler.transform(X_test)

# MLP(multi layered perceptron) classifier

```
-> from sklearn.neural_network import MLPClassifier
   mlp = MLPClassifier(hidden_layer_sizes=(8,), activation='relu',
         batch_size=32, verbose = True, max_iter=10, solver =
         'sgd'
```

# train the model

```
mlp.fit(X_train_std, y_train)
```

# check training score

```
mlp.score(X_train_std, y_train)
```

# check testing score

```
mlp.score(X_test_std, y_test)
```

# compute the prediction

```
y_pred = mlp.predict(X_test_std)
```

# Form confusion matrix, genereate classification report

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
from sklearn.metrics import ConfusionMatrixDisplay


disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
disp.plot()
```

# Form ROC Curve

```
from sklearn import metrics
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_skln)
roc_auc = metrics.auc(fpr, tpr)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
estimator_name="temp")
display.plot()
```

# Linear Regression

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()

xtrain = np.expand_dims(xtrain, 1)  need to increase the
xtest = np.expand_dims(xtest,1)     dim as lr reuqired 2d

lr.fit(xtrain, ytrain)
```

# MLP Regressor

```
from sklearn.neural_network import MLPRegressor

mlp_r = MLPRegressor(hidden_layer_sizes=(10,),
activation='identity', solver='sgd',verbose=True,
max_iter=60)

mlp_r
```

# Naive Bayes Gaussain

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
y_pred = gnb.fit(X_train_std, y_train).predict(X_test_std)
print("Number of mislabeled points out of a total %d points : %d"
 % (X_test_std.shape[0], (y_test != y_pred).sum()))
```

```
gnb.score(X_train_std,y_train)
```

```
0.8662420382165605
```

```
gnb.score(X_test_std,y_test)
```

```
0.9113924050632911
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```