***To read csv from path***

filepath = "D:\\zia\\Datas\\hackathon\\av\\av\_jh\_vehinsCrsSelpred\\"

train = pd.read\_csv(filepath+"train.csv")

test = pd.read\_csv(filepath+"test.csv")

***To find the version***

import sklearn

sklearn.\_\_version\_\_

***To find all the scores in the method***

import sklearn

sorted(sklearn.metrics.SCORERS.keys())

***To compare the features difference between train, test and sample submission files***

display(train.columns.difference(test.columns), test.columns.difference(train.columns),\

test.columns.difference(sample\_sub.columns), sample\_sub.columns.difference(test.columns))

***To concat train and test datasets***

train["source"]="train"

test["source"]="test"

data = pd.concat([train,test])

***Making id column as index***

train.set\_index('id',inplace=True)

test.set\_index('id',inplace=True)

***To find value\_counts with index***

pd.value\_counts(out).sort\_index()

***To change index name or column names***

range\_calc.index.names = ['Loan\_Amount\_Term']

range\_calc.columns=['value\_count','min','max','sum','count','mean','median']

***To find null values total in the dataset***

train.isna().sum()

test.isna().sum()

***To list all the rows having null values in the dataset***

data[data.drop(columns=['Loan\_Status']).isna().any(axis=1)]

***To find features having null in the dataset***

data.isna().sum()[data.isna().sum().values>0]

***To find null values in the dataset(%-wise)***

train.isnull().sum()/train.shape[0] \*100

Or

df.isnull().sum()/len(df))\*100

***To find features(all/specific) having null(with %) in the dataset***

[print(data[i].isna().sum(),"(",round((data[i].isnull().sum()/len(data[i]))\*100,2),"%)-->",i) for i in data.columns if data[i].isna().sum()>0]

or

z = ["Loan\_Status\_num","Gender\_num","Married\_num","Self\_Employed\_num","LoanAmount\_log",

'Education\_num', 'Property\_Area\_num', 'ApplicantIncome\_log','CoapplicantIncome\_log']

[print(data[i].isna().sum(),"(",round((data[i].isnull().sum()/len(data[i]))\*100,2),"%)-->",i) for i in data.drop(columns=z).columns if data[i].isna().sum()>0]

***To find duplicates in the dataset***

train.duplicated().value\_counts(), test.duplicated().value\_counts()

***To drop the duplicate records***

train.drop\_duplicates(inplace=True)

test.drop\_duplicates(inplace=True)

***To find unique values in each column***

train.nunique(), test.nunique()

[print(i,data[i].nunique()) for i in data.columns]

***To find columns having less number of unique values***

[print(data[i].value\_counts(),"\n") for i in data.columns if data[i].nunique()<=20]

***To retreive the column types in dataset(individual or grouped)***

print(train.dtypes.unique())

print(train.dtypes[train.dtypes == np.int64].unique())

print(train.dtypes[train.dtypes != np.int64].unique())

print(train.dtypes[train.dtypes == np.float64].unique())

print(train.dtypes[train.dtypes != np.float64].unique())

print(train.dtypes[train.dtypes == np.object].unique())

print(train.dtypes[train.dtypes != np.object].unique())

***To describe the object datatype columns only***

train.describe(exclude=train.dtypes[train.dtypes != np.object].unique())

Or

data.describe(exclude=np.number).T

***To find a specific values occurence***

data[data['Item\_Visibility']==0].count()['Item\_Visibility']

***To segregate categorical and numerical features***

#categorical features

categorical = train.select\_dtypes(include =[np.object])

print("Categorical Features in Train Set:",categorical.shape[1])

#numerical features

numerical= train.select\_dtypes(include =[np.float64,np.int64])

print("Numerical Features in Train Set:",numerical.shape[1])

Or

num\_col = data.select\_dtypes(include=np.number).columns

cat\_col = data.select\_dtypes(exclude=np.number).columns

***To change categorical to Numerical***

data["Gender"] = np.where(data["Gender"]=="Male",1,0)

data["Occupation"].replace({"Other":0,"Salaried":1,"Self\_Employed":2,"Entrepreneur":3},inplace=True)

***To fill null values with median or mode***

print("Median=",data['Item\_Weight'].median())

print("Null=",data['Item\_Weight'].isna().sum())

data['Item\_Weight'].fillna(value=data['Item\_Weight'].median(),inplace=True)

data['Item\_Weight'].median()

print("Null=",data['Item\_Weight'].isna().sum())

Or

print("Mode=",data['Outlet\_Size'].mode()[0])

print("Null=",data['Outlet\_Size'].isna().sum(),"\n", data['Outlet\_Size'].value\_counts())

data['Outlet\_Size'].fillna(value=data['Outlet\_Size'].mode()[0],inplace=True)

print("Null=",data['Outlet\_Size'].isna().sum(),"\n", data['Outlet\_Size'].value\_counts())

***To display correlation as heatmap***

ax = sns.heatmap(corr, cmap="YlGnBu", annot=True)

***To compare in pairs***

sns.pairplot(data)

***To check outliers using boxplot***

plt.boxplot(data['bmi'])

plt.show()

data[['bmi']].describe().T

data.bmi.agg(["count","mean","median"])

***To convert str to int/down-scaling to check correlation***

data["Self\_Employed\_num"] = data["Self\_Employed"].replace({"Yes":1, "No":0})

data["Property\_Area\_num"] = data["Property\_Area"].replace({"Urban":2, "Semiurban":1, "Rural":0})

data["ApplicantIncome\_log"] = np.log1p(data["ApplicantIncome"])

***To check correlation for specific columns***

data.corr()[['registered']].T

Or

sns.heatmap(data.corr()[['Loan\_Status\_num']],annot=True)

***To check correlation for all columns***

plt.figure(figsize=(12,8))

sns.heatmap(data.corr(),annot=True)

plt.show()

***To do univariate analysis for single feature***

data['Education'].value\_counts().plot(kind = "bar")

***To do univariate analysis with subplots for all features***

%matplotlib inline

plt.figure(figsize=(16,30))

k=1

col\_cnt = 3

row\_cnt = int(np.ceil(len(data.columns)/3))

for col in data.columns:

plt.subplot(row\_cnt,col\_cnt,k)

if data[col].nunique()<25:

title='bar'

data[col].value\_counts().plot(kind ='bar')

else:

title='hist'

data[col].value\_counts().plot(kind ='hist')

#plt.title(title+'\_'+col)

plt.ylabel(title+'\_'+col)

k+=1

plt.show()

***CrossTab Bivariate analysis***

married = pd.crosstab(data["Married"],data['Loan\_Status'])

print("type-data[married]:",type(data["Married"]), "type-data[Loan\_Status]:",type(data["Loan\_Status"]) )

display(married)

display(married.sum(1))

display(married.div(cross.sum(1),axis = 0))

***To do Bivariate analysis with target variable***

tgt\_var = train.columns.difference(test.columns).values[0]

for col in data.drop(columns=['Loan\_ID','Loan\_Status']).columns :

if data[col].nunique()<25:

cross = pd.crosstab(data[col],data[tgt\_var])

cross.div(cross.sum(1), axis=0).plot(kind='bar', stacked=True, legend=True)

***To do Bivariate analysis with target variable with subplots for all features***

%matplotlib inline

j=k=0

col\_cnt = 3

j=[idx for idx, col in enumerate(data.drop(columns=['Loan\_ID','Loan\_Status']).columns) if(data[col].nunique()<25) ]

j=len(j)

row\_cnt = int(np.ceil(j/3))

row\_cnt+=1

fig, ax = plt.subplots(ncols=col\_cnt, nrows=row\_cnt, figsize=(16, row\_cnt\*6))

ax = ax.flatten()

tgt\_var = train.columns.difference(test.columns).values[0]

for idx, col in enumerate(data.drop(columns=['Loan\_ID','Loan\_Status']).columns) :

if data[col].nunique()<25 :

cross = pd.crosstab(data[col],data[tgt\_var])

cross.div(cross.sum(1), axis=0).plot(kind='bar', stacked=True, ax=ax[k])

#plt.title(col)

else :

plt.subplot(row\_cnt,col\_cnt,k+1)

plt.scatter(x=data[col], y=data[tgt\_var].notnull())

plt.xlabel(col)

k+=1

plt.show()

***To find relation of a feature group on another feature***

def range\_calc(data,grouped\_col,groupby\_col):

range\_calc = pd.DataFrame(data[grouped\_col].value\_counts().sort\_index())

range\_calc = pd.concat([range\_calc,

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].min()),

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].max()),

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].sum()),

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].count()),

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].mean()),

pd.DataFrame(data.groupby(grouped\_col)[groupby\_col].median())

],axis=1, ignore\_index=False)

range\_calc.index.names = [grouped\_col]

range\_calc.columns=['value\_count','min','max','sum','count','mean','median']

return range\_calc

'''

display(pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.min()))

display(pd.DataFrame(data2.groupby("Loan\_Amount\_Term")["LoanAmount"].min()))

range\_calc = pd.DataFrame(data2["Loan\_Amount\_Term"].value\_counts().sort\_index())

range\_calc = pd.concat([range\_calc,

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.min()),

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.max()),

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.sum()),

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.count()),

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.mean()),

pd.DataFrame(data2.groupby("Loan\_Amount\_Term").LoanAmount.median())

],axis=1, ignore\_index=False)

range\_calc.index.names = ['Loan\_Amount\_Term']

range\_calc.columns=['value\_count','min','max','sum','count','mean','median']

display(range\_calc.head(25))

'''

range\_calc(data2,"Loan\_Amount\_Term","ApplicantIncome")

***To do binning of a feature***

def binning(data, bins, labels):

bin\_data = pd.cut(data2["Loan\_Amount\_Term"], bins=bins, labels=labels)

return bin\_data

bins = [0,120,180,240,300,360,480]

labels = ['0-120','120-180','180-240','240-300','300-360','360-480']

#data2["Loan\_Amount\_Term\_bin"] = pd.cut(data2["Loan\_Amount\_Term"], bins=bins, labels=labels)

data2["Loan\_Amount\_Term\_bin"] = binning(data2["Loan\_Amount\_Term"],bins,labels)

display(data2["Loan\_Amount\_Term\_bin"].value\_counts().sort\_index())

display(data2["Loan\_Amount\_Term"].value\_counts().sort\_index())

***To do oversampling of the target feature(below not for independent feature)***

# Class count

print('Before Random over-sampling:')

print(train\_preprocessed['Loan\_Status\_num'].value\_counts())

count\_class\_0, count\_class\_1 = train\_preprocessed['Loan\_Status\_num'].value\_counts().sort\_index()

# Divide by class

df\_class\_0 = train\_preprocessed[train\_preprocessed['Loan\_Status\_num'] == 0]

df\_class\_1 = train\_preprocessed[train\_preprocessed['Loan\_Status\_num'] == 1]

#Random oversampling

df\_class\_0\_over = df\_class\_0.sample(count\_class\_1, replace=True)

train\_preprocessed\_ovr = pd.concat([df\_class\_1, df\_class\_0\_over], axis=0)

print('After Random over-sampling:')

print(train\_preprocessed\_ovr.Loan\_Status\_num.value\_counts())

**Note**: do binning before using sampling

***TBD***

pd.options.display.max\_columns = None

pd.set\_option("display.max\_columns", None)

df.sort\_values(by=['Item\_Identifier\_2','Item\_Visibility'])

df["Outlet\_Age"] = datetime.now().year-df["Outlet\_Establishment\_Year"]

pd.Series(model.coef\_,index=test\_X.columns).plot(kind = "bar")

polyreg=make\_pipeline(PolynomialFeatures(degree),LinearRegression())

X[["sex","smoker","region"]] = X[["sex","smoker","region"]].astype("category")

# Assign the cat values

X["sex\_cat"] = X["sex"].cat.codes

X["smoker\_cat"] = X["smoker"].cat.codes

X["region\_cat"] = X["region"].cat.codes

X.head(10)

Item\_Type\_weight = dict(data.groupby(['Item\_Type']).Item\_Weight.mean())

Item\_Type\_weight

data\_copy['Item\_Weight'].fillna(data\_copy['Item\_Type'].map(Item\_Type\_weight),inplace=True)

data['Item\_Weight'].fillna(data['Item\_Type'].map(Item\_Type\_weight),inplace=True)

Did this for item weight mean

data2.isna().sum().reset\_index(name="count").query("count > 0")

***Error***

**TypeError**: '<' not supported between instances of 'str' and 'float'

Soln: Check index or the feature having different data and trying to sort them