Importing library

```
import warnings
warnings.filterwarnings("ignore")
```

```
#import liberaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline
pd.set_option("display.max_columns", None)
```

Read Application

```
app_data=pd.read_csv("application_data.csv")
```

```
app_data.head()
```

Data insepection on Apllication DataSet

```
app_data.info()
```

Data Quality check

check for the percentage null values in Application Dataset

```
pd.set_option("display.max_rows",200)
app_data.isnull().mean() * 100
```

#conclusion: columns with null value more than 47% may give wrong insight hence will drop them

Dropping columns with missing values greater than 47%

```
percentage = 47
threshold = int(((100-percentage)/100)*app_data.shape[0] + 1)
#app_df = app_data.dropna(axis=1, )
app_df = app_data.dropna(axis=1,thresh=threshold)
app_df.head()
```

```
app_df.shape
```

```
app_df.isnull().mean() * 100
```

Imputing Missing value

check the missing value in application datasetb before imputing

```
app_df.info()
```

OCCUPATION_TYPE column has 31% missing values, since its a categorical column, imputing the missing values with a unknown or others values

```
app_df.OCCUPATION_TYPE.isnull().mean()*100
```

```
app_df.OCCUPATION_TYPE.value_counts(normalize=True)*100
```

```
app_df.OCCUPATION_TYPE.fillna("Others",inplace=True)
```

```
app_df.OCCUPATION_TYPE.isnull().mean()*100
```

```
app_df.OCCUPATION_TYPE.value_counts(normalize=True)*100
```

```
# EXT_SOURCE_3 column has 19% missing values
```

```
app_df.EXT_SOURCE_3.isnull().mean()*100
```

```
app_df.EXT_SOURCE_3.value_counts(normalize=True)*100
```

```
app_df.EXT_SOURCE_3.describe()
```

```
sns.boxplot(x=app_df['EXT_SOURCE_3'])
plt.show()
```

#-Conlusion: Since as a numerical columns with no outiers and there is not much difference between Mean and Median. Hence we can impute with Mean or Median

```
app_df['EXT_SOURCE_3'].fillna(app_df['EXT_SOURCE_3'].median(), inplace=True)
```

```
app_df.EXT_SOURCE_3.isnull().mean()*100
```

```
app_df.EXT_SOURCE_3.value_counts(normalize=True)*100
```

```
null_cols=list(app_df.columns[app_df.isna().any()])
len(null_cols)
```

```
app_df.isnull().mean()*100
```

- Handling Missing values in Columns with 13% null values

```
app_df.AMT_REQ_CREDIT_BUREAU_HOUR.value_counts(normalize=True)*100
```

```
app_df.AMT_REQ_CREDIT_BUREAU_DAY.value_counts(normalize=True)*100
```

#- Conclusion : We couls see that 99% of values in the columns AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY AMT_REQ_CREDIT_BUREAU_WEEKAMT_REQ_CREDIT_BUREAU_MON AMT_REQ_CREDIT_BUREAU_QRT AMT_REQ_CREDIT_BUREAU_YEAR is 0.0. Hence impute these columns with the mode

```
cols = [ "AMT_REQ_CREDIT_BUREAU_HOUR",
        "AMT_REQ_CREDIT_BUREAU_DAY",
        "AMT_REQ_CREDIT_BUREAU_WEEK",
        "AMT_REQ_CREDIT_BUREAU_MON", "AMT_REQ_CREDIT_BUREAU_QRT",
        "AMT_REQ_CREDIT_BUREAU_YEAR"]
```

```
for col in cols:
    app_df[col].fillna(app_df[col].mode()[0],inplace=True)
```

```
app_df.isnull().mean()*100
```

Handling Missing values less than 1%

```
null_cols=list(app_df.columns[app_df.isna().any()])
len(null_cols)
```

```
app_df.OBS_30_CNT_SOCIAL_CIRCLE.value_counts(normalize=True)*100
```

```
app_df.EXT_SOURCE_2.value_counts(normalize=True)*100
```

app_df.OBS_30_CNT_SOCIAL_CIRCLE.value_counts(normalize=True)*100

-Conclusion:

-for categorical Columns, impute the missing values with mode

-for numerical columns, imputing missing vlaues with median

app_df.NAME_TYPE_SUITE.fillna(app_df.NAME_TYPE_SUITE .mode()[0], inplace=True)

app_df.CNT_FAM_MEMBERS.fillna(app_df.CNT_FAM_MEMBERS .mode()[0], inplace=True)

#Imputing Numerical Columns

app_df.EXT_SOURCE_2.fillna(app_df.EXT_SOURCE_2 .median(), inplace=True)

app_df.AMT_GOODS_PRICE.fillna(app_df.AMT_GOODS_PRICE .median(), inplace=True)

app_df.AMT_ANNUITY.fillna(app_df.AMT_ANNUITY .median(), inplace=True)

app_df.DEF_60_CNT_SOCIAL_CIRCLE.fillna(app_df.DEF_60_CNT_SOCIAL_CIRCLE .median(), inplace=True)

app_df.DEF_30_CNT_SOCIAL_CIRCLE.fillna(app_df.DEF_30_CNT_SOCIAL_CIRCLE .median(), inplace=True)

app_df.OBS_30_CNT_SOCIAL_CIRCLE.fillna(app_df.OBS_30_CNT_SOCIAL_CIRCLE .median(), inplace=True)

app_df.OBS_60_CNT_SOCIAL_CIRCLE.fillna(app_df.OBS_60_CNT_SOCIAL_CIRCLE .median(), inplace=True)

app_df.DAYS_LAST_PHONE_CHANGE.fillna(app_df.DAYS_LAST_PHONE_CHANGE .median(), inplace=True)

null_cols = list(app_df.columns[app_df.isna().any()])

len(null_cols)

app_df.isnull().mean()*100

Convert Negative values to positive in days variable so that median is nit affected

app_df.DAYS_BIRTH=app_df.DAYS_BIRTH.apply(lambda x: abs(x))

```
app_df.DAYS_EMPLOYED=app_df.DAYS_EMPLOYED.apply(lambda x: abs(x))

app_df.DAYS_ID_PUBLISH=app_df.DAYS_ID_PUBLISH.apply(lambda x: abs(x))

app_df.DAYS_LAST_PHONE_CHANGE=app_df.DAYS_LAST_PHONE_CHANGE.apply(lambda x: abs(x))

app_df.DAYS_REGISTRATION=app_df.DAYS_REGISTRATION.apply(lambda x: abs(x))


# Binning of Continuous variable


# Standing Days columns in variable in Years for easy binning


app_df["YEARS_BIRTH"]= app_df.DAYS_BIRTH.apply(lambda x: int(x//356))

app_df["YEARS_ELPLOYED"]=app_df.DAYS_EMPLOYED.apply(lambda x: int(x//356))

app_df["YEARS_REGISTRATION"]=app_df.DAYS_REGISTRATION.apply(lambda x: int(x//356))

app_df["YEARS_ID_PUBLISH"]= app_df.DAYS_ID_PUBLISH.apply(lambda x: int(x//356))

app_df["YEARS_LAST_PHONE_CHANGE"]= app_df.DAYS_LAST_PHONE_CHANGE.apply(lambda x:
int(x//56))


app_df.AMT_CREDIT.value_counts(normalize=True)*100


app_df.AMT_CREDIT.describe()


app_df["AMT_CREDIT_Category"]=pd.cut(app_df.AMT_CREDIT,
[0,200000,400000,600000,800000,1000000],
                         labels=["Very low Credit","Low Credit","Medium Credit","High Credit","Very High
Credit"])


app_df.AMT_CREDIT_Category.value_counts(normalize=True)*100


app_df["AMT_CREDIT_Category"].value_counts(normalize=True).plot.bar()

plt.show()
```

```
app_df['AGE_Category'] = pd.cut(app_df['YEARS_BIRTH'], [0, 25, 45, 65, 85],
                    labels=["Below 25", "25-45", "45-65", "65-85"])

app_df.AGE_Category.value_counts(normalize=True)*100

app_df['AGE_Category'].value_counts(normalize=True).plot.pie(autopct = '%1.2f%%')
plt.show()

app_df.head()

# Diving Application Dataset with Target Variable as 0 and 1

tar_0 = app_df[app_df.TARGET ==0]
tar_1 = app_df[app_df.TARGET ==1]

app_df.TARGET.value_counts(normalize =True)*100

# Univariate Analysis

cat_cols=list(app_df.columns[app_df.dtypes == object])
num_cols=list(app_df.columns[app_df.dtypes == np.int64])+ list(app_df.columns[app_df.dtypes == np.float64])

cat_cols
```

```
num_cols
```

```
for col in cat_cols:
    print(app_df[col].value_counts(normalize = True))
    plt.figure(figsize=[5,5])
    app_df[col].value_counts(normalize = True).plot.pie(labeldistance = None, autopct='%1.2f%%')
    plt.legend()
```

Plot on Numberical Columns

Categorizing columns with and without flags

```
num_cols_withoutflag = []
num_cols_withflag = []
for col in num_cols:
    if col.startswith("FLAG"):
        num_cols_withflag.append(col)
    else:
        num_cols_withoutflag.append(col)
```

```
num_cols_withflag
```

```
num_cols_withoutflag
```

```
for col in num_cols_withoutflag:
    print(app_df[col].describe())
    plt.figure(figsize = [8,5])
    sns.boxplot(data=app_df,x=col)
    plt.show()
    print("----------------------------------------")
```

```
### Univariate Analysis on Columns with Target 0 to 1

for col in cat_cols:
    print(f"plot on {col} for Target o to 1")
    plt.figure(figsize=[10,7])
    plt.subplot(1,2,1)
    tar_0[col].value_counts(normalize=True).plot.bar()
    plt.title("Target 0")
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.subplot(1,2,2)
    tar_1[col].value_counts(normalize = True).plot.bar()
    plt.title("Target 1")
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.show()
    print("\n----------------------------------------------------------------------------------------------------\n")


### Analysis on AMT_GOODS_PRICE on Traget 0 to 1

plt.figure(figsize=(10,6))
sns.distplot(tar_0['AMT_GOODS_PRICE'],label = 'tar_0',hist=False)
sns.distplot(tar_1['AMT_GOODS_PRICE'],label ='tar_1',hist=False)
plt.legend()
plt.show()
```

Bivariate and Multivariate Analysis

Bivariate Analysis between WEEKDAY_APPR_PROCESS_START VS HOUR_APPR_PROCESS_START

```
plt.figure(figsize=(15,10))

plt.subplot(1,2,1)
sns.boxplot(x='WEEKDAY_APPR_PROCESS_START', y='HOUR_APPR_PROCESS_START',data = tar_0)
plt.subplot(1,2,2)
sns.boxplot(x='WEEKDAY_APPR_PROCESS_START', y='HOUR_APPR_PROCESS_START',data=tar_1)
plt.show()
```

Bivariate Analysis between AGE_CATEGORY vs AMT_CREDIT

```
plt.figure(figsize=(15,10))

plt.subplot(1,2,1)
sns.boxplot(x='AGE_Category', y='AMT_CREDIT',data = tar_0)
plt.subplot(1,2,2)
sns.boxplot(x='AGE_Category', y='AMT_CREDIT',data=tar_1)
plt.show()
```

Pair Plot of Amount Columns for Target 0

```
sns.pairplot(tar_0[["AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE"]]
)
plt.show()
```

Pair Plot of Amount Columns for Targt 1

sns.pairplot(tar_1[["AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE"]]
)

plt.show()

Co-relation between Numerical Columns

corr_data=app_df[["AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE",

"YEARS_BIRTH","YEARS_ELPLOYED","YEARS_REGISTRATION","YEARS_ID_PUBLISH","YEARS_LAST_PH
ONE_CHANGE"]]

corr_data.head()

corr_data.corr()

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

sns.heatmap(corr_data.corr(),annot=True,cmap="RdYlGn")

plt.show()

Split the Numberical variable based on Traget 0 and 1 to find the co_relation

corr_data_0=tar_0[["AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE"
,

"YEARS_BIRTH","YEARS_ELPLOYED","YEARS_REGISTRATION","YEARS_ID_PUBLISH","YEARS_LAST_PH
ONE_CHANGE"]]

corr_data_0.head()

corr_data_1=tar_1[["AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","AMT_GOODS_PRICE"
,

"YEARS_BIRTH","YEARS_ELPLOYED","YEARS_REGISTRATION","YEARS_ID_PUBLISH","YEARS_LAST_PH
ONE_CHANGE"]]

corr_data_1.head()

```
plt.figure(figsize=(10,10))
sns.heatmap(corr_data_0.corr(),annot=True,cmap="RdYlGn")
plt.show()
```

```
plt.figure(figsize=(10,10))
sns.heatmap(corr_data_1.corr(),annot=True,cmap="RdYlGn")
plt.show()
```

Read Previous Application CSV

```
papp_data=pd.read_csv("previous_application-1.csv")
papp_data.head()
```

Data inspection on Previous Application dataset

Get info and shape on the dataset

```
papp_data.info()
```

```
papp_data.shape
```

Data quality Check

Check for Percentage null values in Application dataset

```
papp_data.isnull().mean()*100
```

```
percentage = 49
```

```
threshold_p = int(((100-percentage)/100)*papp_data.shape[0] + 1)
```

```
papp_df = papp_data.dropna(axis=1,thresh=threshold_p)
papp_df.head()
```

```
papp_df.shape
```

Impute Missing values

Check the dtype of missing values in Application dataset before imputing values

```
for col in papp_df.columns:
    if papp_df[col].dtypes == np.int64 or papp_df[col].dtypes == np.float64:
        papp_df[col]=papp_df[col].apply(lambda x:abs(x))
```

Validate if any null values present in dataset

```
null_cols= list(papp_df.columns[papp_df.isna().any()])
len(null_cols)
```

```
papp_df.isnull().mean()*100
```

Binnig od continuous variable
Binning AMT_CREDIT Column

```
papp_df.AMT_CREDIT.describe()
```

```
papp_df["AMT_CREDIT_Category"]=pd.cut(papp_df.AMT_CREDIT,
[0,200000,400000,600000,800000,1000000],
                                   labels=["Very low Credit","Low Credit","Medium Credit","High Credit","Very High
Credit"])
```

```
papp_df["AMT_CREDIT_Category"].value_counts(normalize=True).plot.bar()
plt.show()


papp_df['AMT_GOODS_PRICE_Category'] = pd.qcut(
   papp_df.AMT_GOODS_PRICE,q=[0, 0.25, 0.45, 0.65, 0.85, 1],
   labels=["Very low Price", "Low Price", "Medium Price", "High Price", "Very High Price"]
)


papp_df['AMT_GOODS_PRICE_Category'].value_counts(normalize=True).plot.pie(autopct='%1.2f%%'
)
plt.legend()
plt.show()


# Data Imbalance Check

### Dividing Application Dataset with NAME_CONTRACT_STATUS

approved = papp_df[papp_df.NAME_CONTRACT_STATUS == "Approved"]
cancelled = papp_df[papp_df.NAME_CONTRACT_STATUS == "Canceled"]
refused = papp_df[papp_df.NAME_CONTRACT_STATUS == "Refused"]
unused = papp_df[papp_df.NAME_CONTRACT_STATUS == "Unused offer"]

papp_df.NAME_CONTRACT_STATUS.value_counts(normalize =True)*100

papp_df.NAME_CONTRACT_STATUS.value_counts(normalize=True).plot.pie(autopct='%1.2f%%')
plt.legend()
plt.show()
```

Univariate Analysis

```
cat_cols = list(papp_df.columns[papp_df.dtypes == object])

num_cols = list(papp_df.columns[papp_df.dtypes == np.int64])+ list(papp_df.columns[papp_df.dtypes == np.float64])
```

```
cat_cols
```

```
num_cols
```

```
cat_cols=["NAME_CONTRACT_TYPE","WEEKDAY_APPR_PROCESS_START","NAME_CONTRACT_STATUS","NAME_PAYMENT_TYPE","NAME_SELLER_INDUSTRY","CHANNEL_TYPE","NAME_YIELD_GROUP","PRODUCT_COMBINATION"]
```

```
num_cols=["HOUR_APPR_PROCESS_START","DAYS_DECISION","AMT_ANNUITY","AMT_APPLICATION","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]
```

Plot on Categorical Columns

```
for col in cat_cols:
    print(papp_df[col].value_counts(normalize =True)*100)
    plt.figure(figsize=[5,5])
    papp_df[col].value_counts(normalize =True).plot.pie(labeldistance = None , autopct = '%1.2f%%')
    plt.legend()
    plt.show()
    print("-------------------------------------------------------------------------------------")
```

Plot on Numerical Columns

```
for col in num_cols:
```

```
    print("99th Percentile",np.percentile(papp_df[col],99))
    print(papp_df[col].describe())
    plt.figure(figsize=[10,6])
    sns.boxplot(data=papp_df,x=col)
    plt.show()
    print("-------------------------------------------------------------------------------------")
```

Bivariate and Multivariate Analysis

Bivariate Analysis between WEEKDAY_APPR_PROCESS_START VS AMT_APPLICATION

```
plt.figure(figsize=[10,5])
sns.barplot(x='WEEKDAY_APPR_PROCESS_START',y='AMT_APPLICATION',data=approved)
plt.title("Plot for Approved")
plt.show()

plt.figure(figsize=[10,5])
sns.barplot(x='WEEKDAY_APPR_PROCESS_START',y='AMT_APPLICATION',data=refused)
plt.title("Plot for refused")
plt.show()

plt.figure(figsize=[10,5])
sns.barplot(x='WEEKDAY_APPR_PROCESS_START',y='AMT_APPLICATION',data=unused)
plt.title("Plot for unused")
plt.show()
```

Bivariant Analysis between AMT_ANNUITY vs AMT_GOODS_PRICE

```
plt.figure(figsize=(15,10))
plt.subplot(1,4,1)
plt.title("Approved")
```

```
sns.scatterplot(x='AMT_ANNUITY',y='AMT_GOODS_PRICE', data=approved)

plt.subplot(1,4,2)

plt.title("Cancelled")

sns.scatterplot(x='AMT_ANNUITY',y='AMT_GOODS_PRICE', data=cancelled)

plt.subplot(1,4,3)

plt.title("Refused")

sns.scatterplot(x='AMT_ANNUITY',y='AMT_GOODS_PRICE', data=refused)

plt.subplot(1,4,4)

plt.title("Unused")

sns.scatterplot(x='AMT_ANNUITY',y='AMT_GOODS_PRICE', data=unused)

plt.show()
```

Co-relation between Numerical columns

```
corr_approved=approved[["DAYS_DECISION","AMT_ANNUITY","AMT_APPLICATION","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]]

corr_refused=refused[["DAYS_DECISION","AMT_ANNUITY","AMT_APPLICATION","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]]

corr_cancelled=cancelled[["DAYS_DECISION","AMT_ANNUITY","AMT_APPLICATION","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]]

corr_unused=unused[["DAYS_DECISION","AMT_ANNUITY","AMT_APPLICATION","AMT_CREDIT","AMT_GOODS_PRICE","CNT_PAYMENT"]]
```

Co-relation for Numerical columns for Approved

```
plt.figure(figsize=[10,10])

sns.heatmap(corr_approved.corr(),annot=True,cmap="Blues")

plt.title("Heat Map plot for approved")

plt.show()
```

Co-relation for Numerical columns for Refused

```
plt.figure(figsize=[10,10])
sns.heatmap(corr_refused.corr(),annot=True,cmap="Blues")
plt.title("Heat Map plot for Refused")
plt.show()
```

Co-relation for Numerical columns for Cancelled

```
plt.figure(figsize=[10,10])
sns.heatmap(corr_cancelled.corr(),annot=True,cmap="Blues")
plt.title("Heat Map plot for Cancelled")
plt.show()
```

Co-relation for Numerical columns for Unused

```
plt.figure(figsize=[10,10])
sns.heatmap(corr_unused.corr(),annot=True,cmap="Blues")
plt.title("Heat Map plot for Unused")
plt.show()
```

Merge the Application and Previous Application DataFrames

```
merge_df = app_df.merge(papp_df,on=["SK_ID_CURR"],how='left')
merge_df.head()
```

```
merge_df.info()
```

Filtering required columns for our Analysis

```
for col in merge_df.columns:
    if col.startswith("FLAG"):
        merge_df.drop(columns=col, axis=1,inplace=True)
```

```
merge_df.shape
```

```
res1 = pd.pivot_table(data=merge_df, index=["NAME_INCOME_TYPE","NAME_CLIENT_TYPE"],
columns=["NAME_CONTRACT_STATUS"],values="TARGET", aggfunc="mean")
res1
```

```
plt.figure(figsize=[10,10])
sns.heatmap(res1,annot=True,cmap='BuPu')
plt.show()
```

```
res2 = pd.pivot_table(data=merge_df, index=["CODE_GENDER","NAME_SELLER_INDUSTRY"],
                 columns=["TARGET"], values="AMT_GOODS_PRICE_x", aggfunc='sum')
```

```
res2
```

```
plt
.figure(figsize=[12,15])
sns.heatmap(res2,annot=True,cmap='BuPu')
plt.show()
```