Import All necessity Functions

```
##### Import all necessity functions for Machine Learning #####
import sys
import math
import string
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as shc
import warnings
import zipfile
import cv2
import os
import random
from collections import Counter
from functools import reduce
from itertools import chain
from google.colab.patches import cv2_imshow
from keras.preprocessing import image
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, GridSearchCV
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.feature_selection import mutual_info_classif, mutual_info_regression, SelectK
from imblearn.under_sampling import RandomUnderSampler, NearMiss
from imblearn.over_sampling import RandomOverSampler, SMOTE, SMOTEN, SMOTENC, SVMSMOTE, KM
from imblearn.ensemble import EasyEnsembleClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor, NearestNeighbors
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier, SGDR
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, ExtraTreeClassifie
from sklearn.ensemble import BaggingClassifier, BaggingRegressor, RandomForestClassifier,
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor, GradientBoostingClassi
from sklearn.metrics import classification_report, mean_absolute_error, mean_squared_error
from xgboost import XGBClassifier, XGBRegressor

##### Download keras #####
!pip install keras

##### Remove all warnings #####
import warnings
```

```
warnings.filterwarnings("ignore")

##### Import all necessity functions for Neural Network #####
import tensorflow as tf
from keras.models import Sequential, Model
from keras.utils import plot_model
from keras.layers import Dense, Conv2D, LSTM, GRU, RNN, Flatten, AvgPool2D, MaxPool2D, Glo
from keras.activations import tanh, relu, sigmoid, softmax, swish
from keras.regularizers import L1, L2, L1L2
from keras.optimizers import SGD, Adagrad, Adadelta, RMSprop, Adam, Adamax, Nadam
from keras.initializers import HeNormal, HeUniform, GlorotNormal, GlorotUniform
from keras.losses import SparseCategoricalCrossentropy, CategoricalCrossentropy, hinge, MS
import keras.utils as image
from google.colab.patches import cv2_imshow
from keras.utils import plot_model

##### Plotting the confusion matrix #####
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import confusion_matrix
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub]
Requirement already satisfied: keras in /usr/local/lib/python3.9/dist-packages (2.11.0)
```

Import the dataset

```
df = pd.read_csv('/content/train.csv')
df.head()
```

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location
1	C-33999	America	M	32	4952.91	Low	Working	NaN	Semi-

Determine the number of records of this dataset

```
def find_feature_record(df):
  if df.empty:
    print('The dataset is empty'.captialize())
  else:
    print('The # of records of this dataset is {}'.format(df.shape[0]))
    print('The # of columns of this dataset is {}'.format(df.shape[1]))

try:
  find_feature_record(df)
except Exception as e:
  print(e.with_traceback)
```

```
The # of records of this dataset is 30000
The # of columns of this dataset is 24
```

Determine the shape of this dataset

```
def shape_dataset(dataset):
  if dataset.empty:
    print('The dataset is empty'.captialize())
  else:
    print('The shape of this dataset is {}'.format(dataset.shape))

try:
  shape_dataset(df)
except Exception as e:
  print(e.with_traceback)
```

```
The shape of this dataset is (30000, 24)
```

Determine how many NaN value presence and their column name

```
def check_nan(df):
  if df.empty:
    print('The dataset is empty'.captialize())
  else:
    return df.isnull().sum()[df.isnull().sum() > 0]

```

```
try:
  nan_value = check_nan(df)
except Exception as e:
  print(e.with_traceback)
else:
  for feature, value in zip(nan_value.index, nan_value.values):
    print('{} feature has NaN value # {}'.format(feature, value),'\n')
    print('*'*40)

  print('The total NaN value is in this dataset is = {}'.format(df.isnull().sum().sum()))
```

```
Gender feature has NaN value # 53

****************************************
Income (USD) feature has NaN value # 4576

****************************************
Income Stability feature has NaN value # 1683

****************************************
Type of Employment feature has NaN value # 7270

****************************************
Current Loan Expenses (USD) feature has NaN value # 172

****************************************
Dependents feature has NaN value # 2493

****************************************
Credit Score feature has NaN value # 1703

****************************************
Has Active Credit Card feature has NaN value # 1566

****************************************
Property Age feature has NaN value # 4850

****************************************
Property Location feature has NaN value # 356

****************************************
Loan Sanction Amount (USD) feature has NaN value # 340

****************************************
The total NaN value is in this dataset is = 25062
```

Determine those features have NaN value are categorial or Neumrical

```
def find_categorical_numerical(df):
  if df.empty:
    print('The dataset is empty'.capitalize())
  else:
```

```
    return df.dtypes[df.dtypes.index.isin(df.isnull().sum()[df.isnull().sum() > 0].index)]

try:
  finding = find_categorical_numerical(df)
except Exception as e:
  print(e.with_traceback)
else:
  for feature, value in zip(finding.index, finding.values):
    print('{} feature datatype is # {}'.format(feature, value))
    print('-'*50)
```

```
Gender feature datatype is # object
--------------------------------------------------
Income (USD) feature datatype is # float64
--------------------------------------------------
Income Stability feature datatype is # object
--------------------------------------------------
Type of Employment feature datatype is # object
--------------------------------------------------
Current Loan Expenses (USD) feature datatype is # float64
--------------------------------------------------
Dependents feature datatype is # float64
--------------------------------------------------
Credit Score feature datatype is # float64
--------------------------------------------------
Has Active Credit Card feature datatype is # object
--------------------------------------------------
Property Age feature datatype is # float64
--------------------------------------------------
Property Location feature datatype is # object
--------------------------------------------------
Loan Sanction Amount (USD) feature datatype is # float64
--------------------------------------------------
```

Split the dataset into Categorical and Numerical for analysing

```
def split_cat_numerical(cat = None, num = None):
  if cat == 'categorical':
    categorical_df = df[df.dtypes[df.dtypes == 'object'].index]
    return categorical_df
  elif(num == 'numerical'):
     numerical_df = df[df.dtypes[df.dtypes != 'object'].index]
     return numerical_df
  else:
    raise Exception('Splitting cannot be done.'.captialize())

try:
  cat_df = split_cat_numerical(cat = 'categorical', num = None)
  num_df = split_cat_numerical(cat = None, num = 'numerical')
except Exception as e:
  print(e.with_traceback)
else:
```

```
16 else:
17   print('done'.capitalize())
```

```
Done
```

```
cat_df.head()
```

	Customer ID	Name	Gender	Income Stability	Profession	Type of Employment	Location	Expense Type 1	Exp Ty
0	C-36995	Frederica Shealy	F	Low	Working	Sales staff	Semi-Urban	N	
1	C-33999	America Calderone	M	Low	Working	NaN	Semi-Urban	N	
2	C-3770	Rosetta Verne	F	High	Pensioner	NaN	Semi-Urban	N	

```
num_df.head()
```

	Age	Income (USD)	Loan Amount Request (USD)	Current Loan Expenses (USD)	Dependents	Credit Score	No. of Defaults	Property ID	Property Age	P
0	56	1933.05	72809.58	241.08	3.0	809.44	0	746	1933.05	
1	32	4952.91	46837.47	495.81	1.0	780.40	0	608	4952.91	
2	65	988.19	45593.04	171.95	1.0	833.15	0	546	988.19	
3	65	NaN	80057.92	298.54	2.0	832.70	1	890	NaN	
4	31	2614.77	113858.89	491.41	NaN	745.55	1	715	2614.77	

EDA - Exploratory data analysis

Delete the Customer ID and Name from the dataset

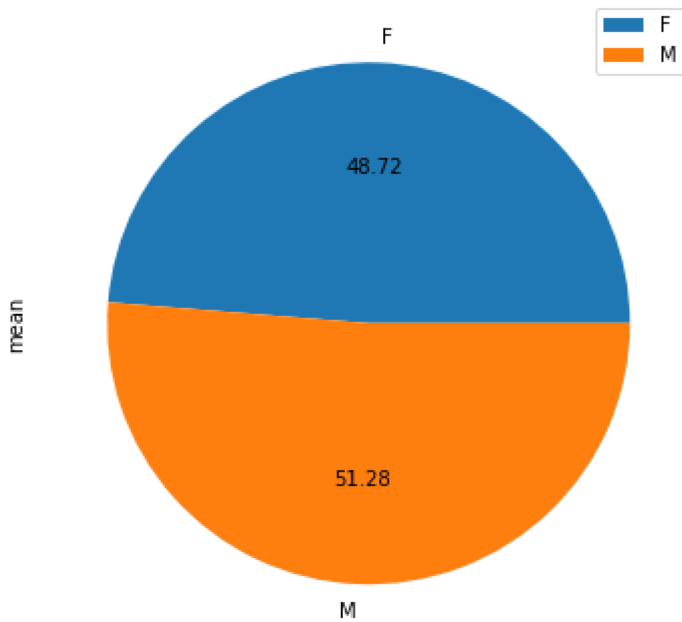
```
def remove_features(features = None):
  if len(features) == 1:
    df.drop(features, axis = 1, inplace = True)
  else:
    for column in features:
      df.drop(column, axis = 1, inplace = True)
      print('{} column is deleted from the dataset.'.capitalize().format(column))

```

```
try:
  remove_features(['Customer ID', 'Name'])
except Exception as e:
  print(e.with_traceback)
```

```
Customer ID column is deleted from the dataset.
Name column is deleted from the dataset.
```

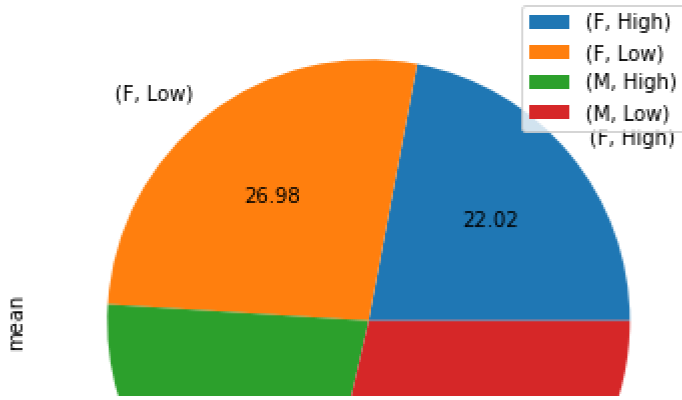
The distribution of average income based on Gender

```
df.groupby(['Gender'])['Income (USD)'].\
                      agg(['mean']).plot(kind = 'pie',\
                      subplots = True, autopct='%.2f', figsize = (8, 6))
plt.show()
```



The distribution of average income based on Gender and Income Stability

```
df.groupby(['Gender', 'Income Stability'])['Income (USD)'].\
                                          agg(['mean']).plot(kind = 'pie',\
                                          subplots = True, autopct='%.2f', figsize = (8, 6
plt.show()
```



Check, what's the maximum, minimum, mean `Loan Amount Request (USD)` based on the `Profession`


```
df.groupby(['Profession'])['Loan Amount Request (USD)'].\
                           agg(['max', 'min', 'mean']).\
                           sort_values(by = 'mean', ascending = False)
```

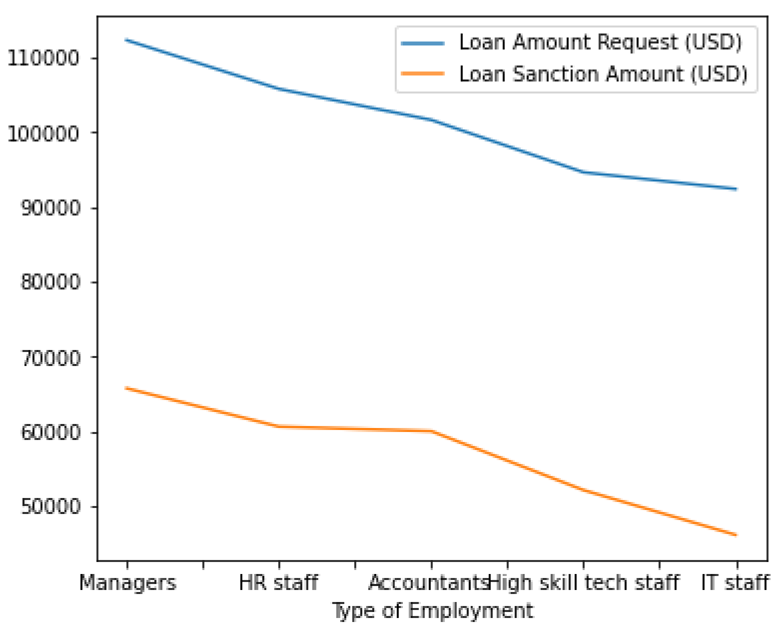
Profession	max	min	mean
Maternity leave	108967.56	108967.56	108967.560000
State servant	417936.91	6310.26	97543.778331
Commercial associate	602384.15	6174.70	96521.471468
Pensioner	387288.84	6498.75	86344.841022
Businessman	94747.20	74449.41	84598.305000
Working	621497.82	6048.24	84391.484042
Unemployed	89038.91	77097.81	83068.360000
Student	58056.11	58056.11	58056.110000

Find out, the max-5 `Type of Employment` based on their `Loan Amount Request (USD)`

```
df.groupby(['Type of Employment'])['Loan Amount Request (USD)'].mean().\
                                                           sort_values(ascending = Fal
                                                           plot(kind = 'barh', figsize
plt.show()
```

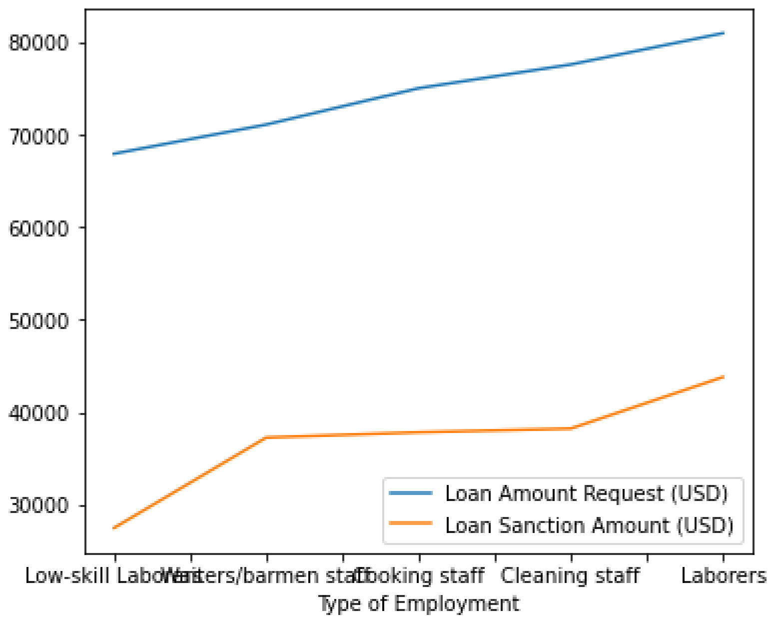
Find out, the mean of requesting `Loan Amount` and getting `Loan sanction` based on the top5 `Type of Employement`

```
top_5 = df.groupby(['Type of Employment'])['Loan Amount Request (USD)'].mean().\
                                          sort_values(ascending = False).head(5).reset_ind

index = df.groupby(['Type of Employment'])['Loan Amount Request (USD)'].mean().\
                                                               sort_values(ascending = Fal

dummy_df = df[df.loc[:, 'Type of Employment'].isin(index)]

top_5['Loan Sanction Amount (USD)'] = dummy_df.groupby(['Type of Employment'])['Loan Sanct
                                                     sort_values(ascending = False).head(

top_5.set_index('Type of Employment').plot(figsize = (6, 5))
plt.show()
```




```
top_5 = df.groupby(['Type of Employment'])['Loan Amount Request (USD)'].mean().\
                                          sort_values(ascending = True).head(5).reset_inde
```

```

index = df.groupby(['Type of Employment'])['Loan Amount Request (USD)'].mean().\
                                                                 sort_values(ascending = Tru

dummy_df = df[df.loc[:, 'Type of Employment'].isin(index)]

top_5['Loan Sanction Amount (USD)'] = dummy_df.groupby(['Type of Employment'])['Loan Sanct
                                                         sort_values(ascending = True).head(5

top_5.set_index('Type of Employment').plot(figsize = (6, 5))
plt.show()
```

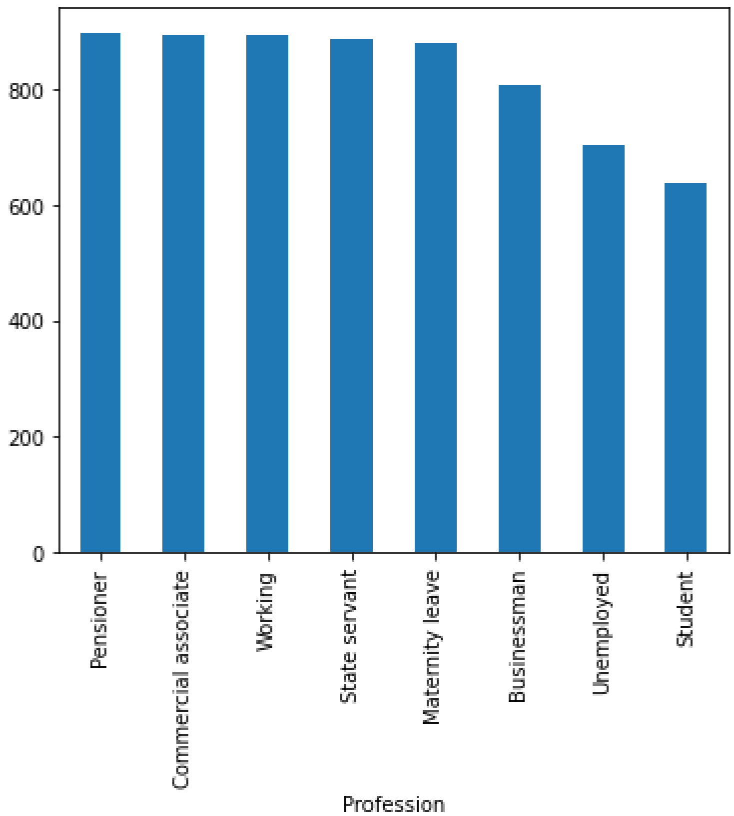


Show, the relationship of `Income Stability`, `location` with the features like `Loan amount` and `Loan Sanction`

```
pd.pivot_table(index = 'Income Stability', columns = ['Location'],\
               values = ['Loan Amount Request (USD)', 'Loan Sanction Amount (USD)'],\
               aggfunc = ['mean'], data = df).plot(kind = 'pie', subplots = True, figsize
plt.show()
```

Find out, the highest `credit score` based on their `Profession`

```
df.groupby(['Profession'])['Credit Score'].max().sort_values(ascending = False).plot(kind
plt.show()
```



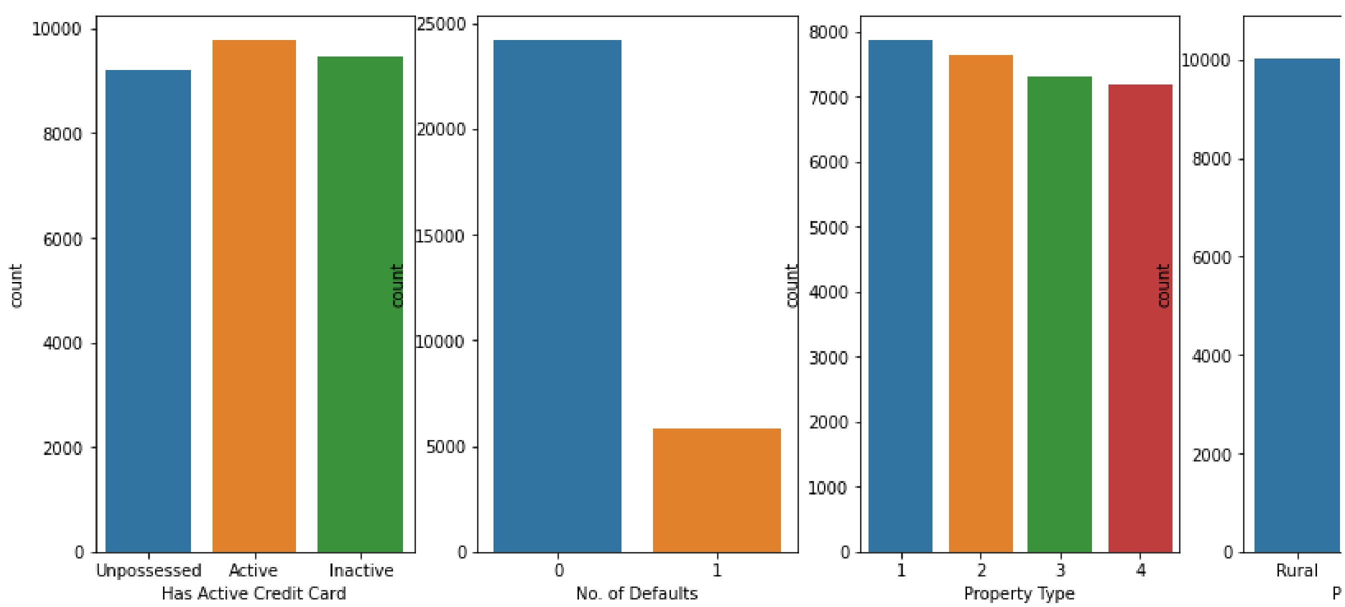
Find out, the top `Credit Score` using `Profession` & show their mean, max, min, count of `Type of Employement` and their `Loan request` and `Loan Sanction`

```
def find_queries(group):
  return group['Credit Score'].max()

df[df.loc[:, 'Profession'].isin(df.groupby(['Profession']).apply(find_queries).sort_values
                              loc[:, ['Profession', 'Loan Amount Request (USD)', 'Loan Sanctio
                              groupby(['Profession']).agg(['min', 'max', 'mean']).head()
```

	Loan Amount Request (USD)			Loan Sanction Amount (USD)		
	min	max	mean	min	max	mean
Profession						
Businessman	74449.41	94747.20	84598.305000	59559.53	66323.04	62941.2850
Maternity leave	108967.56	108967.56	108967.560000	76277.29	76277.29	76277.2900

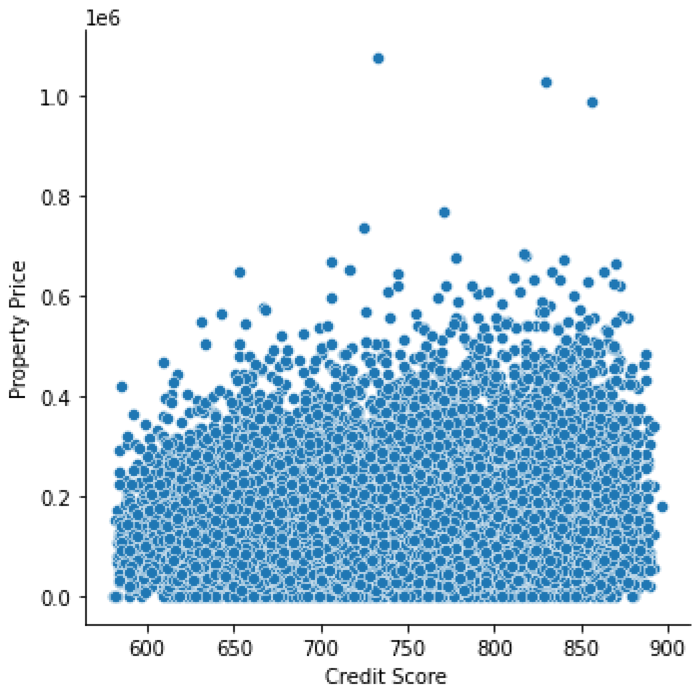
Show the countplot of the `Has Active Credit Card` column

```
plt.figure(figsize = (16, 6))
plt.subplot(1, 4, 1)
sns.countplot(df.loc[:, 'Has Active Credit Card'])

plt.subplot(1, 4, 2)
sns.countplot(df.loc[:, 'No. of Defaults'])

plt.subplot(1, 4, 3)
sns.countplot(df.loc[:, 'Property Type'])

plt.subplot(1, 4, 4)
sns.countplot(df.loc[:, 'Property Location'])
plt.show()
```



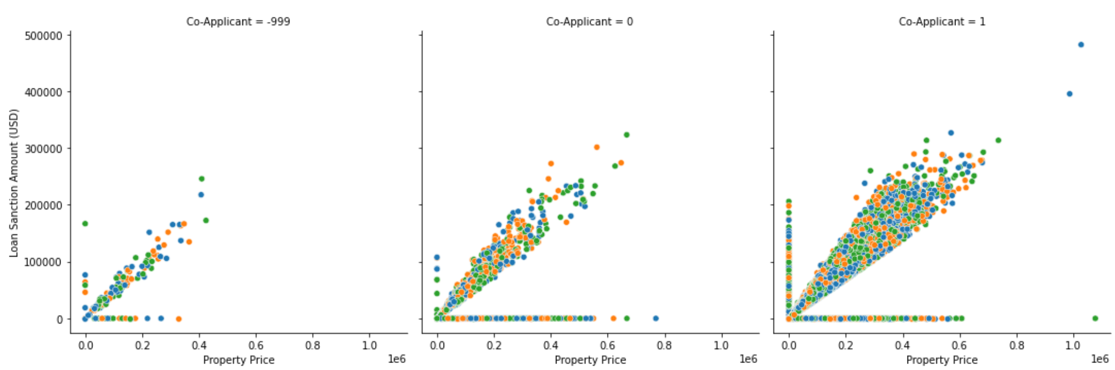
Show the realtionship of `Credit Score` and `Property Price`

```
sns.relplot(data = df, x = 'Credit Score', y = 'Property Price', kind = 'scatter')
plt.show()
```



Find out, the relationship of `Property location`, `Property price` and `Loan Sanction`

```
1 sns.relplot(x = 'Property Price', y = 'Loan Sanction Amount (USD)', hue = 'Property Locati
2 plt.show()
```



Find out, the top `Property price` based on `their location`

```
1 df.groupby(['Property Location'])['Property Price'].agg(['max', 'count', 'sum', 'mean'])
```

	max	count	sum	mean
Property Location				
Rural	1028082.64	10041	1.319019e+09	131363.317538
Semi-Urban	1077966.73	10387	1.367362e+09	131641.637175
Urban	678932.62	9216	1.219563e+09	132331.057394

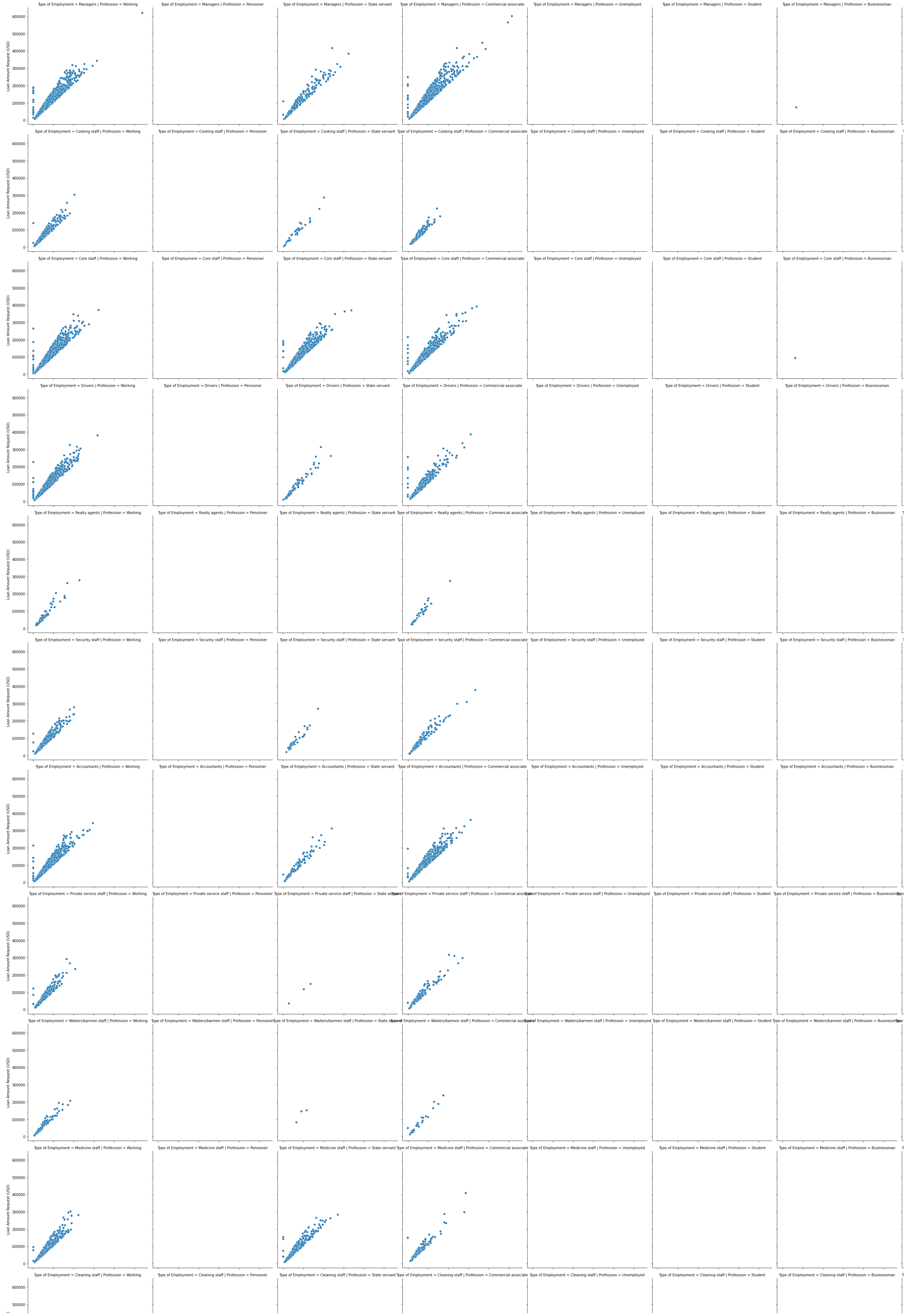
Find out, the mean and count of `Loan request` & `Loan Sanction` based on `property type`

```
pd.pivot_table(index = ['Property Location'], columns = ['Property Type'], values = ['Loan
                                                                                      'Loan
                                                                                       aggf
                                                                                       data
```

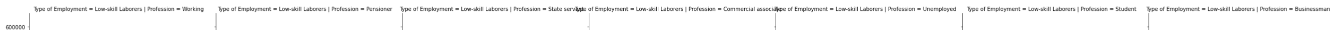
	mean					
	Loan Amount Request (USD)				**Loan Sanction Amount (U**	
Property Type	**1**	**2**	**3**	**4**	**1**	**2**
Property Location						
Rural	88687.272223	87595.814497	88685.207034	88950.360624	47502.564716	47529.9458
Semi-Urban	89271.307594	88719.609821	88428.335978	88298.671714	48131.483707	47081.5727
Urban	88832.400781	87706.488145	90261.747884	90506.063780	47794.297994	46333.3403

Show the realtionship of `Property Price`, `Loan Request`, `Profession`, `Type of Employement`

```
sns.relplot(x = 'Property Price', y = 'Loan Amount Request (USD)', col = 'Profession', row
```

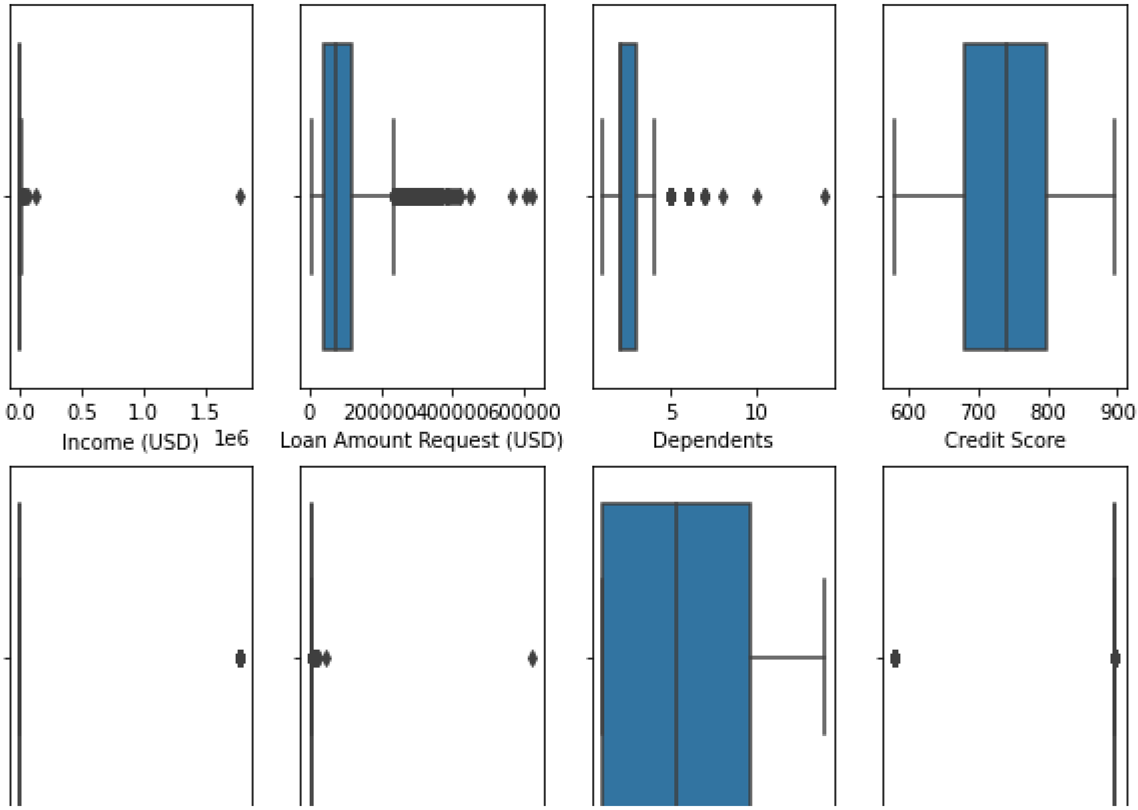

Type of Employment = Managers | Profession = Working
Type of Employment = Managers | Profession = Pensioner
Type of Employment = Managers | Profession = State servant
Type of Employment = Managers | Profession = Commercial associate
Type of Employment = Managers | Profession = Unemployed
Type of Employment = Managers | Profession = Student
Type of Employment = Managers | Profession = Businessman
Type of Employment = Cooking staff | Profession = Working
Type of Employment = Cooking staff | Profession = Pensioner
Type of Employment = Cooking staff | Profession = State servant
Type of Employment = Cooking staff | Profession = Commercial associate
Type of Employment = Cooking staff | Profession = Unemployed
Type of Employment = Cooking staff | Profession = Student
Type of Employment = Cooking staff | Profession = Businessman
Type of Employment = Core staff | Profession = Working
Type of Employment = Core staff | Profession = Pensioner
Type of Employment = Core staff | Profession = State servant
Type of Employment = Core staff | Profession = Commercial associate
Type of Employment = Core staff | Profession = Unemployed
Type of Employment = Core staff | Profession = Student
Type of Employment = Core staff | Profession = Businessman
Type of Employment = Drivers | Profession = Working
Type of Employment = Drivers | Profession = Pensioner
Type of Employment = Drivers | Profession = State servant
Type of Employment = Drivers | Profession = Commercial associate
Type of Employment = Drivers | Profession = Unemployed
Type of Employment = Drivers | Profession = Student
Type of Employment = Drivers | Profession = Businessman
Type of Employment = Realty agents | Profession = Working
Type of Employment = Realty agents | Profession = Pensioner
Type of Employment = Realty agents | Profession = State servant
Type of Employment = Realty agents | Profession = Commercial associate
Type of Employment = Realty agents | Profession = Unemployed
Type of Employment = Realty agents | Profession = Student
Type of Employment = Realty agents | Profession = Businessman
Type of Employment = Security staff | Profession = Working
Type of Employment = Security staff | Profession = Pensioner
Type of Employment = Security staff | Profession = State servant
Type of Employment = Security staff | Profession = Commercial associate
Type of Employment = Security staff | Profession = Unemployed
Type of Employment = Security staff | Profession = Student
Type of Employment = Security staff | Profession = Businessman
Type of Employment = Accountants | Profession = Working
Type of Employment = Accountants | Profession = Pensioner
Type of Employment = Accountants | Profession = State servant
Type of Employment = Accountants | Profession = Commercial associate
Type of Employment = Accountants | Profession = Unemployed
Type of Employment = Accountants | Profession = Student
Type of Employment = Accountants | Profession = Businessman
Type of Employment = Private service staff | Profession = Working
Type of Employment = Private service staff | Profession = Pensioner
Type of Employment = Private service staff | Profession = State servant
Type of Employment = Private service staff | Profession = Commercial associate
Type of Employment = Private service staff | Profession = Unemployed
Type of Employment = Private service staff | Profession = Student
Type of Employment = Private service staff | Profession = Businessman
Type of Employment = Waiters/barmen staff | Profession = Working
Type of Employment = Waiters/barmen staff | Profession = Pensioner
Type of Employment = Waiters/barmen staff | Profession = State servant
Type of Employment = Waiters/barmen staff | Profession = Commercial associate
Type of Employment = Waiters/barmen staff | Profession = Unemployed
Type of Employment = Waiters/barmen staff | Profession = Student
Type of Employment = Waiters/barmen staff | Profession = Businessman
Type of Employment = Medicine staff | Profession = Working
Type of Employment = Medicine staff | Profession = Pensioner
Type of Employment = Medicine staff | Profession = State servant
Type of Employment = Medicine staff | Profession = Commercial associate
Type of Employment = Medicine staff | Profession = Unemployed
Type of Employment = Medicine staff | Profession = Student
Type of Employment = Medicine staff | Profession = Businessman
Type of Employment = Cleaning staff | Profession = Working
Type of Employment = Cleaning staff | Profession = Pensioner
Type of Employment = Cleaning staff | Profession = State servant
Type of Employment = Cleaning staff | Profession = Commercial associate
Type of Employment = Cleaning staff | Profession = Unemployed
Type of Employment = Cleaning staff | Profession = Student
Type of Employment = Cleaning staff | Profession = Businessman
Loan Amount Request (USD)
600000
500000
400000
300000
200000
100000
0

Check there is an outliers or not in the dataset



```
plt.figure(figsize = (10, 12))
plt.subplot(3, 4, 1)
sns.boxplot(df.loc[:, 'Income (USD)'])

plt.subplot(3, 4, 2)
sns.boxplot(df.loc[:, 'Loan Amount Request (USD)'])

plt.subplot(3, 4, 3)
sns.boxplot(df.loc[:, 'Dependents'])

plt.subplot(3, 4, 4)
sns.boxplot(df.loc[:, 'Credit Score'])

plt.subplot(3, 4, 5)
sns.boxplot(df.loc[:, 'No. of Defaults'])

plt.subplot(3, 4, 6)
sns.boxplot(df.loc[:, 'Property Age'])

plt.subplot(3, 4, 7)
sns.boxplot(df.loc[:, 'Property Type'])

plt.subplot(3, 4, 8)
sns.boxplot(df.loc[:, 'Co-Applicant'])

plt.subplot(3, 4, 9)
sns.boxplot(df.loc[:, 'Property Price'])

plt.subplot(3, 4, 10)
sns.boxplot(df.loc[:, 'Loan Sanction Amount (USD)'])
```

```
<AxesSubplot:xlabel='Loan Sanction Amount (USD)'>
```



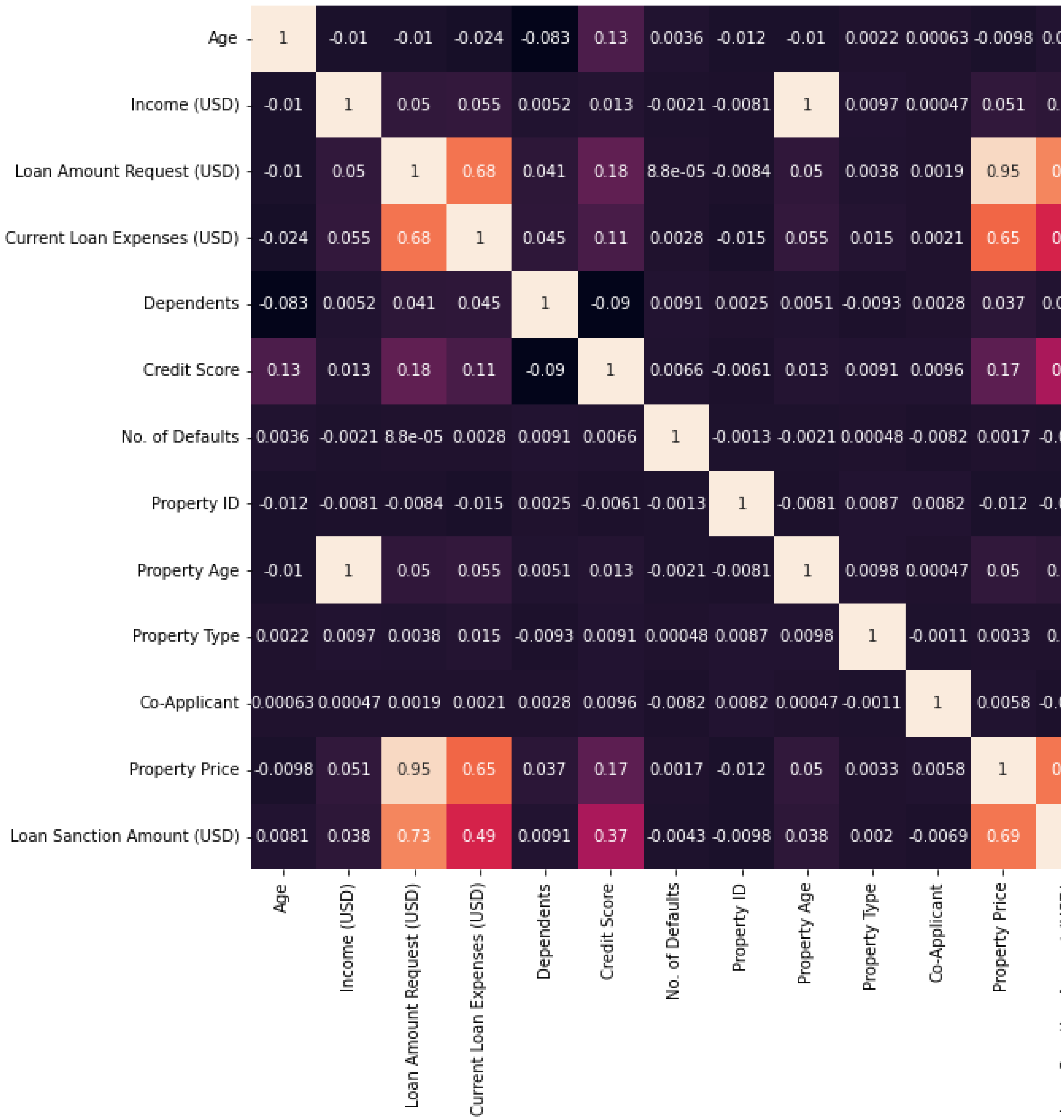
Show the description of this entire dataset

```
1 df.describe()
```

	Age	Income (USD)	Loan Amount Request (USD)	Current Loan Expenses (USD)	Dependents	Credit Score
count	30000.000000	2.542400e+04	30000.000000	29828.000000	27507.000000	28297.000000
mean	40.092300	2.630574e+03	88826.333855	400.936876	2.253027	739.885381
std	16.045129	1.126272e+04	59536.949605	242.545375	0.951162	72.163846
min	18.000000	3.777000e+02	6048.240000	-999.000000	1.000000	580.000000
25%	25.000000	1.650457e+03	41177.755000	247.667500	2.000000	681.880000
50%	40.000000	2.222435e+03	75128.075000	375.205000	2.000000	739.820000
75%	55.000000	3.090593e+03	119964.605000	521.292500	3.000000	799.120000
max	65.000000	1.777460e+06	621497.820000	3840.880000	14.000000	896.260000

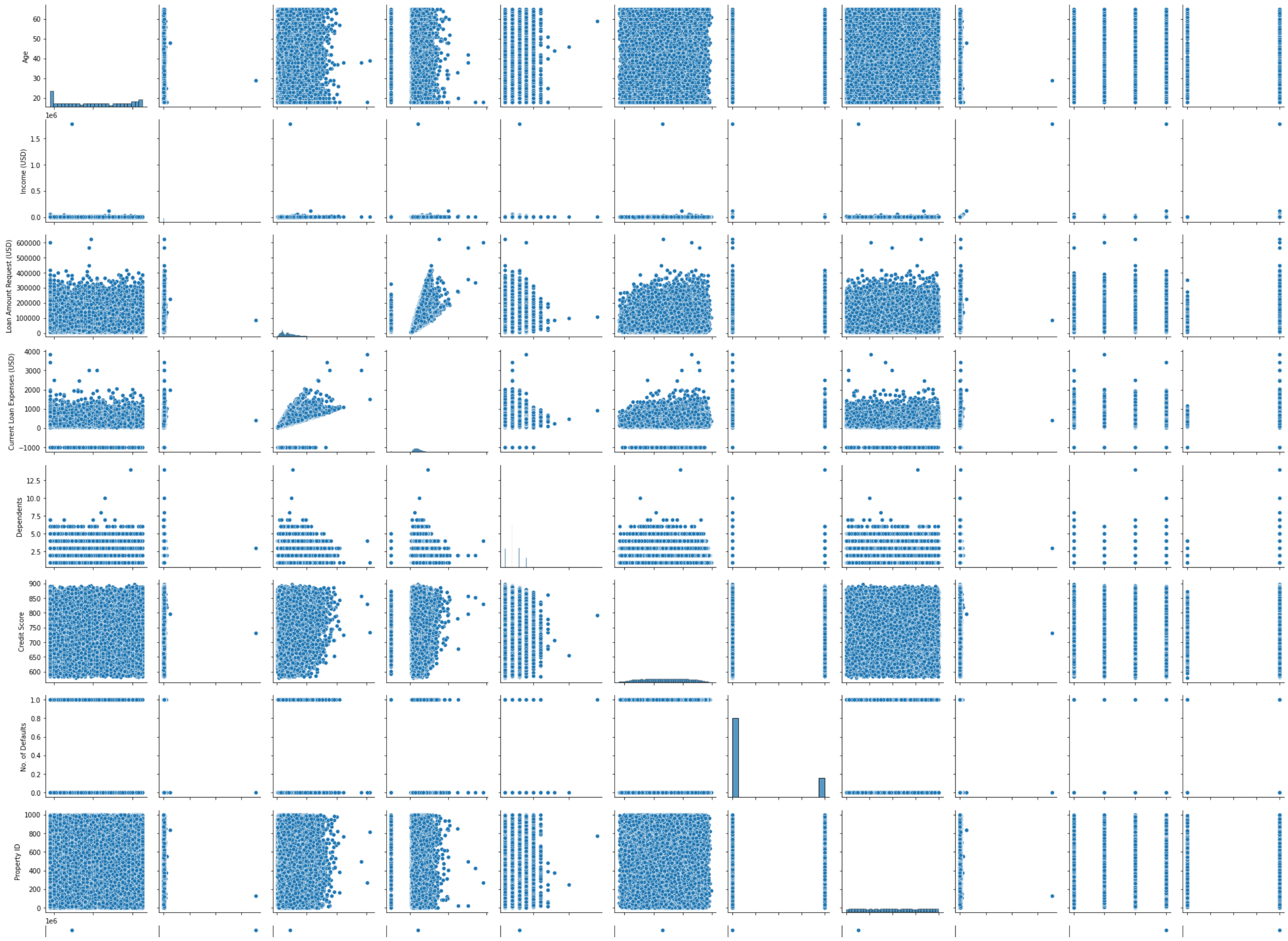
Plot the correlation of this dataset

```
plt.figure(figsize = (12, 10))
sns.heatmap(df.corr(), annot = True)
plt.show()
```



Plot the pairplot of this entire dataset

```
sns.pairplot(df)
plt.show()
```



Feature Engineering for the Dataset

```
Handle the Missing Value

Handle the Categorical Dataset

Splitting the dataset

Normalization Approach
```


Handle the Missing Value


```
#### For ``Gender`` - We will use the new gender called T ####
def NaN_Fill(activate = None, feature = None):
  if activate == 'activated':
    df.loc[:, feature] = np.where(df.loc[:, feature].isnull(), 'T', df.loc[:, feature])
  else:
    raise Exception('Not possible.')

try:
  NaN_Fill(activate = 'activated', feature = 'Gender')
```

```
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

```
Done
```

For `Type of Employment` - We will use the new gender called `homie`

```
def NaN_Fill(activate = None, feature = None):
  if activate == 'activated':
    df.loc[:, feature] = np.where(df.loc[:, feature].isnull(), 'homie', df.loc[:, feature]
  else:
    raise Exception('Not possible.')

try:
  NaN_Fill(activate = 'activated', feature = 'Type of Employment')
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

```
Done
```

For `Loan Sanction Amount (USD)` - We will delete all records

```
def dropna_feature(activate = None, feature = None):
  if activate == 'activated':
    df.dropna(subset = [feature], axis = 0, inplace = True)
  else:
    raise Exception('Not possible.')

try:
  dropna_feature(activate = 'activated', feature = 'Loan Sanction Amount (USD)')
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

```
Done
```

For `Income Stability` - We will use the new gender called `Missing`

```
def NaN_Fill(activate = None, feature = None):
  if activate == 'activated':
    df.loc[:, feature] = np.where(df.loc[:, feature].isnull(), 'Missing', df.loc[:, featur
```

```
  else:
    raise Exception('Not possible.')

try:
  NaN_Fill(activate = 'activated', feature = 'Income Stability')
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

    Done

For `Has Active Credit Card` - We will use the new gender called `Missing`

```
def NaN_Fill(activate = None, feature = None):
  if activate == 'activated':
    df.loc[:, feature] = np.where(df.loc[:, feature].isnull(), 'Missing', df.loc[:, featur
  else:
    raise Exception('Not possible.')

try:
  NaN_Fill(activate = 'activated', feature = 'Has Active Credit Card')
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

    Done

```
try:
  NaN_Fill(activate = 'activated', feature = 'Property Location')
except Exception as e:
  print(e.with_traceback)
else:
  print('Done')
```

    Done

```
# def mean_encoding(activate = None, feature = None):
#   if activate == 'activated':
#     df.loc[:, feature] = df.loc[:, feature].std()
#   else:
#     raise Exception('Not possible')

# try:
#   for features in ['Income (USD)', 'Current Loan Expenses (USD)', 'Dependents', 'Credit
#     mean_encoding(activate = 'activated', feature = features)
# except Exception as e:
```

```
#   print(e.with_traceback)
# else:
#   print('Done')
```

Do the `random sample imputation` for all `Neumerical` features

```
def impute_nan_random_sample(activate = None, column_name_ = None):
  if activate == 'activated':
    index_null_values_ = df.loc[df.loc[:, column_name_].isnull(), :][column_name_].index
    random_values_     = df.loc[:, column_name_].dropna().sample(df.loc[:, column_name_].i
    for value_, index_ in enumerate(df.loc[df.loc[:, column_name_].isnull()][column_name_]
      df.loc[index_, column_name_] = random_values_[value_]


try:
  for features in ['Income (USD)', 'Current Loan Expenses (USD)', 'Dependents', 'Credit Sc
    impute_nan_random_sample(activate = 'activated', column_name_ = features)
except Exception as e:
  print(e.with_traceback)
else:
  print('DONE')
```

```
DONE
```

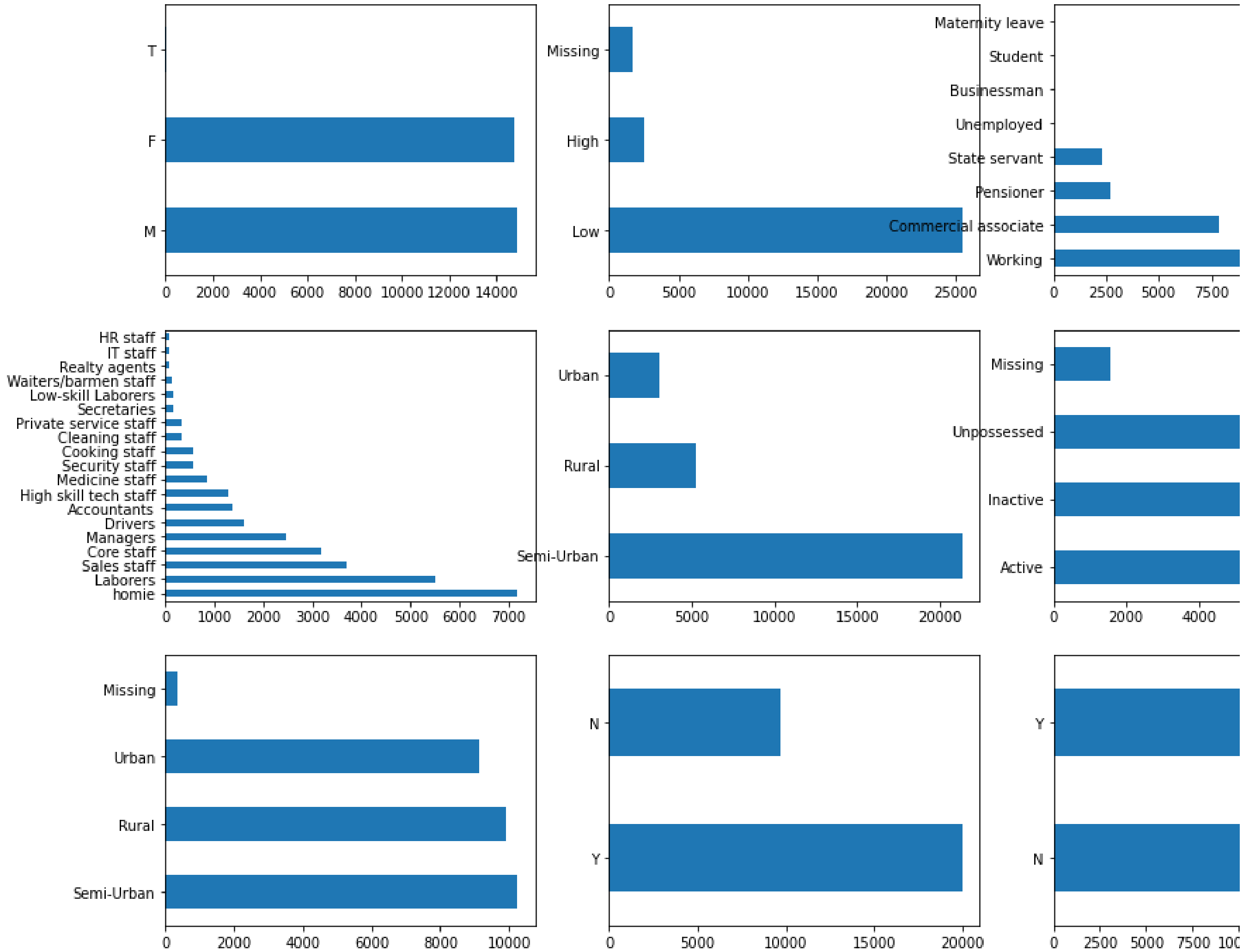
Check `NaN` value presence in this dataset or not

```
if df.isnull().sum().sum() == 0:
  print('There is No NaN value in this dataset.'.capitalize())
else:
  print('There is NaN value in this dataset.'.capitalize())
```

```
There is no nan value in this dataset.
```

Handle the Categorical Data

```
plt.figure(figsize = (16, 12))
for index, feature in enumerate(['Gender', 'Income Stability', 'Profession',\
                                 'Type of Employment', 'Location', 'Has Active Credit Card
                                 'Property Location', 'Expense Type 2', 'Expense Type 1'])
  plt.subplot(3, 3, index + 1)
  df.loc[:, feature].value_counts().plot(kind = 'barh')
```




```
def mean_encoding_all(activate = None, feature = None):
  if activate == 'activated':
    df.loc[:, feature] = df.loc[:, feature].map(df.loc[:, feature].value_counts().to_dict(
  else:
    raise Exception('Not possible.'.capitalize())

try:
  for index, feature_ in enumerate(['Gender', 'Income Stability', 'Profession', 'Type of E
                                    'Location', 'Has Active Credit Card', 'Property Locati
                                    'Expense Type 1', 'Expense Type 2']):
    mean_encoding_all(activate = 'activated', feature = feature_)
except Exception as e:
  print(e.with_traceback)
else:
  print('DONE')
```

```
DONE
```

```
1 df.head()
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)
0	14718	56	1933.05	25458	16739	3698	21317	72809.58	241.08
1	14890	32	4952.91	25458	16739	7188	21317	46837.47	495.81
2	14718	65	988.19	2544	2718	7188	21317	45593.04	171.95
3	14718	65	1918.47	2544	2718	7188	5280	80057.92	298.54
4	14718	31	2614.77	25458	16739	1297	21317	113858.89	491.41

5 rows × 22 columns

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29660 entries, 0 to 29999
Data columns (total 22 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Gender                       29660 non-null  int64  
 1   Age                          29660 non-null  int64  
 2   Income (USD)                 29660 non-null  float64
 3   Income Stability             29660 non-null  int64  
 4   Profession                   29660 non-null  int64  
 5   Type of Employment           29660 non-null  int64  
 6   Location                     29660 non-null  int64  
 7   Loan Amount Request (USD)    29660 non-null  float64
 8   Current Loan Expenses (USD)  29660 non-null  float64
 9   Expense Type 1               29660 non-null  int64  
 10  Expense Type 2               29660 non-null  int64  
 11  Dependents                   29660 non-null  float64
 12  Credit Score                 29660 non-null  float64
 13  No. of Defaults              29660 non-null  int64  
 14  Has Active Credit Card       29660 non-null  int64  
 15  Property ID                  29660 non-null  int64  
 16  Property Age                 29660 non-null  float64
 17  Property Type                29660 non-null  int64  
 18  Property Location            29660 non-null  int64  
 19  Co-Applicant                 29660 non-null  int64  
 20  Property Price               29660 non-null  float64
 21  Loan Sanction Amount (USD)   29660 non-null  float64
dtypes: float64(8), int64(14)
memory usage: 6.2 MB
```

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

Normalizing the dataset

```
minmax_scaler = MinMaxScaler()
X_normalized  = minmax_scaler.fit_transform(df)
df_normalised = pd.DataFrame(X_normalized, columns = df.columns)
df_normalised.head()
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	C Ex
0	0.988408	0.808511	0.000875	1.000000	1.000000	0.509625	1.000000	0.108476	0.
1	1.000000	0.297872	0.002575	1.000000	1.000000	1.000000	1.000000	0.066276	0.
2	0.988408	1.000000	0.000344	0.037227	0.162325	1.000000	1.000000	0.064254	0.
3	0.988408	1.000000	0.000867	0.037227	0.162325	1.000000	0.121453	0.120253	0.
4	0.988408	0.276596	0.001259	1.000000	1.000000	0.172264	1.000000	0.175174	0.

5 rows × 22 columns

Split the dataset into train and test

```
X = df_normalised.iloc[:, :-1]
y = df_normalised.iloc[:, -1]
```

Splitting the dataset into train and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =

print('The shape of X_tarin is = {}'.format(X_train.shape),'\n')
print('The shape of X_test is  = {}'.format(X_test.shape),'\n')
print('The shape of y_train is = {}'.format(y_train.shape),'\n')
print('The shape of y_test is  = {}'.format(y_test.shape),'\n')
```

```
The shape of X_tarin is = (22245, 21)

The shape of X_test is  = (7415, 21)
```

```
The shape of y_train is = (22245,)

The shape of y_test is  = (7415,)
```

Model Building

```
models = {
    'KNN': KNeighborsRegressor(),
    'LR' : LinearRegression(),
    'SVR': SVR(),
    'DT' : DecisionTreeRegressor(),
    'RF' : RandomForestRegressor(),
    'GB' : GradientBoostingRegressor(),
    'XGB': XGBRegressor()
}

MAE, MSE, RMSE, R2 = [], [], [], []

for model_name, model_object in models.items():
  model = model_object
  model.fit(X_train, y_train)

  predicted = model.predict(X_test)

  MAE.append(mean_absolute_error(predicted, y_test))
  MSE.append(mean_squared_error(predicted, y_test))
  RMSE.append(np.sqrt(mean_absolute_error(predicted, y_test)))
  R2.append(r2_score(predicted, y_test))
```

```
model = ['KNN', 'LR', 'SVR', 'DT', 'RF', 'GB', 'XGB']

for model_name, mae, mse, rmse, r2 in zip(model, MAE, MSE, RMSE, R2):
  print('{} MAE is {}, MSE is {}, RMSE is {}, R2 Score is {} '.format(model_name, mae, mse
  print('*'*120)
```

```
KNN MAE is 0.0638534407305653, MSE is 0.007184624663268531, RMSE is 0.25269238360220775,

*****************************************************************************************
LR MAE is 0.04479903060591726, MSE is 0.004227066493187742, RMSE is 0.2116578148945067,

*****************************************************************************************
SVR MAE is 0.06573061933298312, MSE is 0.005911744153942446, RMSE is 0.2563798340996872,

*****************************************************************************************
DT MAE is 0.031672381366825396, MSE is 0.005453135327984628, RMSE is 0.1779673603974206

*****************************************************************************************
```

```
RF MAE is 0.02665696077995848, MSE is 0.002710158228648405, RMSE is 0.16326959539350394,

*************************************************************************************
GB MAE is 0.02848786629850074, MSE is 0.0026038292436277428, RMSE is 0.1687834894132146

*************************************************************************************
XGB MAE is 0.026043110250265854, MSE is 0.0026510122012625963, RMSE is 0.16137877881018

*************************************************************************************
```

Check for `RandomForestRegressor`

```
LR = RandomForestRegressor()
LR.fit(X_train, y_train)

predicted = LR.predict(X_test)

evaluation_df = pd.DataFrame(predicted, columns = ['Predict'])
pd.concat([evaluation_df, pd.DataFrame(y_test.values, columns = ['Actual'])], axis = 1).he
```

	Predict	Actual
0	0.093459	0.115137
1	0.085973	0.106983
2	0.016879	0.020934
3	0.166799	0.002069
4	0.072688	0.081956

Select the `Important Features` from the dataset

Using `Pearson Correlation` Technique

`RandomForestRegressor` used for finding the `feature Importance`

`Pearson Correlation` Technique

```
def correlation(dataset_, threshold_):

    col_corr_ = set()
    corr_matrix = dataset_.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
```

```
            if abs(corr_matrix.iloc[i, j]) > threshold_:
                col_name_ = corr_matrix.columns[i]
                col_corr_.add(col_name_)

    return col_corr_
```

```
correlation(df_normalised, 0.65)
```

```
{'Current Loan Expenses (USD)',
 'Loan Sanction Amount (USD)',
 'Property Age',
 'Property Price'}
```

Use `Random Forest` to find the Feature Importance

```
RF = RandomForestRegressor()
RF.fit(X_train, y_train)

predicted = RF.predict(X_test)
```

```
feature_imporatnce = pd.DataFrame(RF.feature_importances_, columns = ['Feature Importance'
feature_imporatnce.set_index(X.columns).sort_values(by = 'Feature Importance', ascending =
```

	Feature Importance
Loan Amount Request (USD)	0.577052
Credit Score	0.160554
Co-Applicant	0.073832
Income Stability	0.030457
Property Price	0.023473
Property ID	0.022180
Current Loan Expenses (USD)	0.019599
Age	0.017363
Property Age	0.015077
Income (USD)	0.013291

```
df_important = df_normalised.loc[:,feature_imporatnce.set_index(X.columns).\
                  sort_values(by = 'Feature Importance',\
                  ascending = False)[0:7].index]

df_important.head()
```

	Loan Amount Request (USD)	Credit Score	Co-Applicant	Income Stability	Property Price	Property ID
0	0.108476	0.725479	1.000	1.000000	0.112082	0.746493
1	0.066276	0.633656	1.000	1.000000	0.051707	0.608216
2	0.064254	0.800449	0.999	0.037227	0.068065	0.546092
3	0.120253	0.799026	1.000	0.037227	0.113480	0.890782
4	0.175174	0.523462	1.000	1.000000	0.194229	0.715431

Split the dataset into `independent` & `dependent` feature with `Feature Selection - Random Forest`

```
X_important = df_important
```

```
X_train, X_test, y_train, y_test = train_test_split(X_important, y, test_size = 0.25, rand

print('The shape of X_tarin is = {}'.format(X_train.shape),'\n')
print('The shape of X_test is  = {}'.format(X_test.shape),'\n')
print('The shape of y_train is = {}'.format(y_train.shape),'\n')
print('The shape of y_test is  = {}'.format(y_test.shape),'\n')
```

```
The shape of X_tarin is = (22245, 7)

The shape of X_test is  = (7415, 7)

The shape of y_train is = (22245,)

The shape of y_test is  = (7415,)
```

Model Building

```
models = {
    'KNN': KNeighborsRegressor(),
    'LR' : LinearRegression(),
    'SVR': SVR(),
    'DT' : DecisionTreeRegressor(),
    'RF' : RandomForestRegressor(),
    'GB' : GradientBoostingRegressor(),
    'XGB': XGBRegressor()
}

MAE, MSE, RMSE, R2 = [], [], [], []

```

```
for model_name, model_object in models.items():
  model = model_object
  model.fit(X_train, y_train)

  predicted = model.predict(X_test)

  MAE.append(mean_absolute_error(predicted, y_test))
  MSE.append(mean_squared_error(predicted, y_test))
  RMSE.append(np.sqrt(mean_absolute_error(predicted, y_test)))
  R2.append(r2_score(predicted, y_test))
```

```
model = ['KNN', 'LR', 'SVR', 'DT', 'RF', 'GB', 'XGB']

for model_name, mae, mse, rmse, r2 in zip(model, MAE, MSE, RMSE, R2):
  print('{} MAE is {}, MSE is {}, RMSE is {}, R2 Score is {} '.format(model_name, mae, mse
  print('*'*120)
```

```
KNN MAE is 0.03991594081086272, MSE is 0.0042570827170690315, RMSE is 0.19978974150557

****************************************************************************************
LR MAE is 0.044869966903444185, MSE is 0.00422960827789803, RMSE is 0.21182532167671594,

****************************************************************************************
SVR MAE is 0.06227702469271442, MSE is 0.0054030433318580555, RMSE is 0.24955365093044

****************************************************************************************
DT MAE is 0.0308598733433046, MSE is 0.005161709599147438, RMSE is 0.1756697849469413, 

****************************************************************************************
RF MAE is 0.02609631532287773, MSE is 0.002724331998128325, RMSE is 0.16154354002211826,

****************************************************************************************
GB MAE is 0.0284549828004946, MSE is 0.002611718785359302, RMSE is 0.1686860480315269, 

****************************************************************************************
XGB MAE is 0.02612359193914408, MSE is 0.0027299448892521166, RMSE is 0.16162794294039

****************************************************************************************
```

Check for `RandomForestRegressor`

```
random_forest = XGBRegressor()
random_forest.fit(X_train, y_train)

predicted = random_forest.predict(X_test)

evaluation_df = pd.DataFrame(predicted, columns = ['Predict'])
pd.concat([evaluation_df, pd.DataFrame(y_test.values, columns = ['Actual'])], axis = 1).he
```

	Predict	Actual
0	0.104898	0.115137
1	0.092464	0.106983
2	0.020697	0.020934
3	0.162695	0.002069
4	0.071119	0.081956

Using `Stacking Regressor` to evaluate the model

```
estimators = [
    ('GB', GradientBoostingRegressor()),
    ('XGB', XGBRegressor())
]

StackingRegressor_ = StackingRegressor(estimators = estimators, final_estimator = XGBRegre
StackingRegressor_.fit(X_train, y_train)
predicted = StackingRegressor_.predict(X_test)

print('MAE is # {}'.format(mean_absolute_error(predicted, y_test)))
print('MSE is # {}'.format(mean_squared_error(predicted, y_test)))
print('RMSE is # {}'.format(np.mean(mean_squared_error(predicted, y_test))))
print('R2 is # {}'.format(r2_score(predicted, y_test)))

evaluation_df = pd.DataFrame(predicted, columns = ['Predict'])
pd.concat([evaluation_df, pd.DataFrame(y_test.values, columns = ['Actual'])], axis = 1).he
```

```
MAE is # 0.02565574788523634
MSE is # 0.0027161683929956154
RMSE is # 0.0027161683929956154
R2 is # 0.6548782872725348
```

	Predict	Actual
0	0.102465	0.115137
1	0.098963	0.106983
2	0.022414	0.020934
3	0.155950	0.002069
4	0.068526	0.081956

Use `KFold - 5` cross Validation for this Selection Feature

```
KFold_ = KFold(n_splits = 5, shuffle = True, random_state = 42)
```

```

MAE_, MSE_, RSME_, R2_, count = [], [], [], [], 1

for train_index, test_index in KFold_.split(X):
  print('# of Cross Validation {} is runnung'.format(count))

  X_train, X_test = X.iloc[train_index, :], X.iloc[test_index,:]
  y_train, y_test = y[train_index], y[test_index]

  StackingRegressor_ = StackingRegressor(estimators = estimators, final_estimator = XGBReg
  StackingRegressor_.fit(X_train, y_train)
  predicted = StackingRegressor_.predict(X_test)

  MAE_.append(mean_absolute_error(predicted, y_test))
  MSE_.append(mean_squared_error(predicted, y_test))
  RSME_.append(np.sqrt(mean_squared_error(predicted, y_test)))
  R2_.append(r2_score(predicted, y_test))

  count = count + 1
```

```
# of Cross Validation 1 is runnung
# of Cross Validation 2 is runnung
# of Cross Validation 3 is runnung
# of Cross Validation 4 is runnung
# of Cross Validation 5 is runnung
```

```
print('The mean MAE is  = {}'.format(np.array(MAE_).mean(),'\n'))
print('The mean MSE is  = {}'.format(np.array(MSE_).mean(),'\n'))
print('The mean RSME is = {}'.format(np.array(RSME_).mean(),'\n'))
print('The mean R2 is   = {}'.format(np.array(R2_).mean(),'\n'))
```

```
The mean MAE is  = 0.02549977605079265
The mean MSE is  = 0.002614186959615045
The mean RSME is = 0.05109530251813684
The mean R2 is   = 0.6647505097983358
```

Using PCA- Principle component Analysis

```
# PCA_ = PCA(n_components = None)
# PCA_.fit_transform(X)
```

```
# print('The Explained variance ratio is given below.\n')
# np.cumsum(PCA_.explained_variance_ratio_)
```

```
PCA_ = PCA(n_components = 10)
X_trans = PCA_.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_trans, y, test_size = 0.25, random_s

print('The shape of X_tarin is = {}'.format(X_train.shape),'\n')
print('The shape of X_test is  = {}'.format(X_test.shape),'\n')
print('The shape of y_train is = {}'.format(y_train.shape),'\n')
print('The shape of y_test is  = {}'.format(y_test.shape),'\n')
```

```
The shape of X_tarin is = (22245, 21)

The shape of X_test is  = (7415, 21)

The shape of y_train is = (22245,)

The shape of y_test is  = (7415,)
```

```
models = {
    'KNN': KNeighborsRegressor(),
    'LR' : LinearRegression(),
    'SVR': SVR(),
    'DT' : DecisionTreeRegressor(),
    'RF' : RandomForestRegressor(),
    'GB' : GradientBoostingRegressor(),
    'XGB': XGBRegressor()
}

MAE, MSE, RMSE, R2 = [], [], [], []

for model_name, model_object in models.items():
  model = model_object
  model.fit(X_train, y_train)

  predicted = model.predict(X_test)

  MAE.append(mean_absolute_error(predicted, y_test))
  MSE.append(mean_squared_error(predicted, y_test))
  RMSE.append(np.sqrt(mean_absolute_error(predicted, y_test)))
  R2.append(r2_score(predicted, y_test))
```

```
model = ['KNN', 'LR', 'SVR', 'DT', 'RF', 'GB', 'XGB']

for model_name, mae, mse, rmse, r2 in zip(model, MAE, MSE, RMSE, R2):
  print('{} MAE is {}, MSE is {}, RMSE is {}, R2 Score is {} '.format(model_name, mae, mse
  print('*'*120)
```

```
KNN MAE is 0.0638534407305653, MSE is 0.007184624663268531, RMSE is 0.25269238360220775,

*****************************************************************************************
LR MAE is 0.044799030605917264, MSE is 0.004227066493187741, RMSE is 0.2116578148945067

*****************************************************************************************
```