LoanTap Regression Project

1.Define Problem Statement and perform Exploratory Data Analysis

The Goal of this project is to predict credit rating of customers to mitigate the risk of defaults and make precise evaluations of customer creditworthiness.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency, f_oneway, ttest_ind,spearmanr
from scipy.stats import chi2, shapiro, boxcox, ttest_rel
from scipy.stats import chisquare, kruskal
import statsmodels.api as sm
import scipy.stats as stats
```

In [358]:

```
df=pd.read_csv('C:/Users/rahul.kumar/Downloads/Credit.csv')
```

In [359]:

```
df.head()
```

Out[359]:

	Unnamed: 0	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	1	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	2	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	3	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	4	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	5	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331

In [360]:

```
df.shape
```

Out[360]:

```
(400, 12)
```

In [361]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  400 non-null    int64  
 1   Income      400 non-null    float64
 2   Limit       400 non-null    int64  
 3   Rating      400 non-null    int64  
 4   Cards       400 non-null    int64  
 5   Age         400 non-null    int64  
 6   Education   400 non-null    int64  
 7   Gender      400 non-null    object
```

```
 8   Student      400 non-null    object
 9   Married      400 non-null    object
 10  Ethnicity    400 non-null    object
 11  Balance      400 non-null    int64
dtypes: float64(1), int64(7), object(4)
memory usage: 37.6+ KB
```

In [362]:

```
df.isna().sum()
```

Out[362]:

```
Unnamed: 0    0
Income        0
Limit         0
Rating        0
Cards         0
Age           0
Education     0
Gender        0
Student       0
Married       0
Ethnicity     0
Balance       0
dtype: int64
```

In [363]:

```
# drop serial no.
df=df.iloc[:,1:]
```

In [364]:

```
df
```

Out[364]:

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331
...	...	...	...	...	...	...	...	...	...	...	...
395	12.096	4100	307	3	32	13	Male	No	Yes	Caucasian	560
396	13.364	3838	296	5	65	17	Male	No	No	African American	480
397	57.872	4171	321	5	67	12	Female	No	Yes	Caucasian	138
398	37.728	2525	192	1	44	13	Male	No	Yes	Caucasian	0
399	18.701	5524	415	5	64	7	Female	No	No	Asian	966

400 rows × 11 columns

In [365]:

```
df.describe(include='all')
```

Out[365]:

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400	400	400	400	40
unique	NaN	NaN	NaN	NaN	NaN	NaN	2	2	2	3	

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	
top	NaN	NaN	NaN	NaN	NaN	NaN	Female	No	Yes	Caucasian	
freq	NaN	NaN	NaN	NaN	NaN	NaN	207	360	245	199	
mean	45.218885	4735.600000	354.940000	2.957500	55.667500	13.450000	NaN	NaN	NaN	NaN	52
std	35.244273	2308.198848	154.724143	1.371275	17.249807	3.125207	NaN	NaN	NaN	NaN	45
min	10.354000	855.000000	93.000000	1.000000	23.000000	5.000000	NaN	NaN	NaN	NaN	
25%	21.007250	3088.000000	247.250000	2.000000	41.750000	11.000000	NaN	NaN	NaN	NaN	6
50%	33.115500	4622.500000	344.000000	3.000000	56.000000	14.000000	NaN	NaN	NaN	NaN	45
75%	57.470750	5872.750000	437.250000	4.000000	70.000000	16.000000	NaN	NaN	NaN	NaN	86
max	186.634000	13913.000000	982.000000	9.000000	98.000000	20.000000	NaN	NaN	NaN	NaN	199

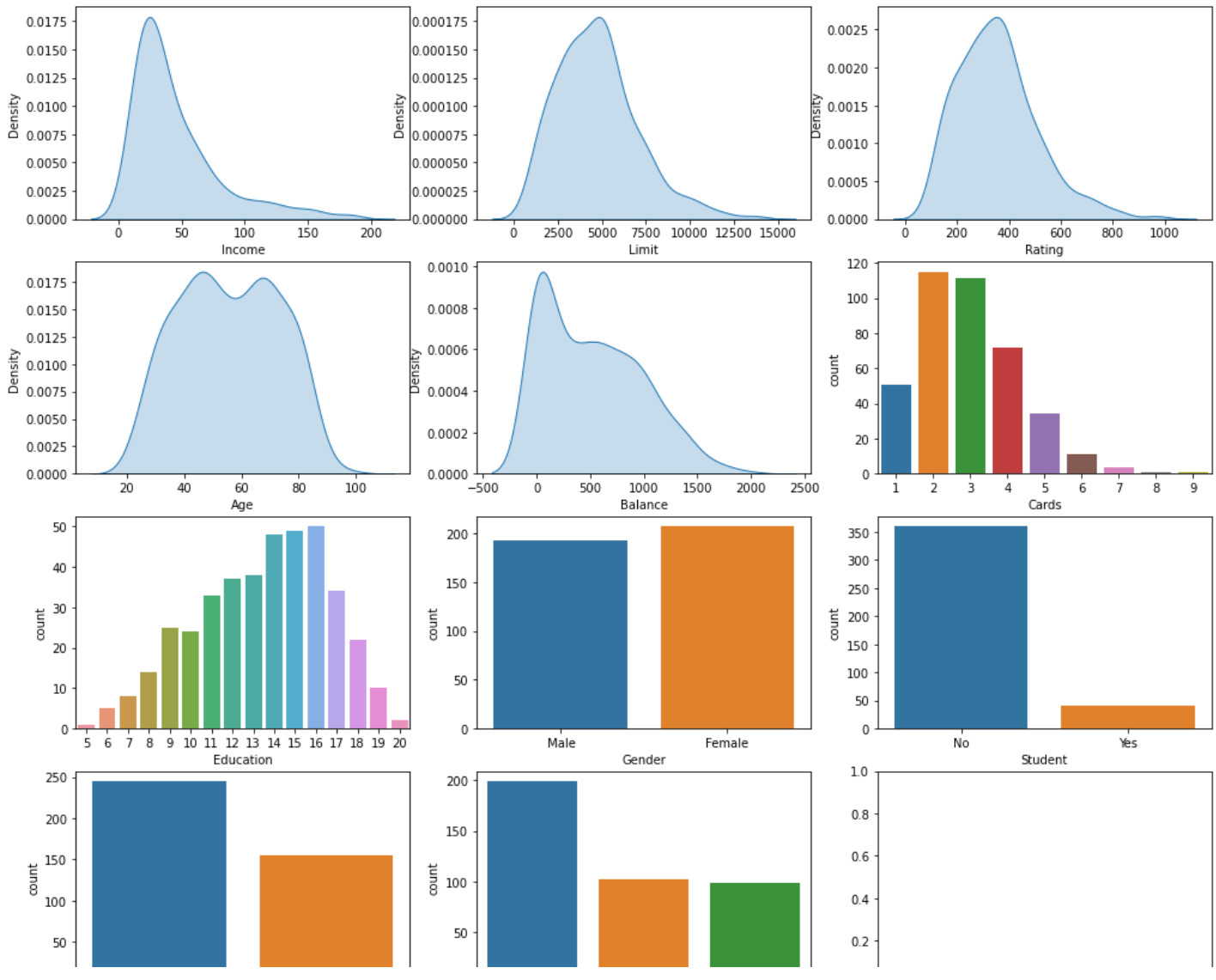
In [366]:

```
fig, axes = plt.subplots(4, 3, figsize=(17, 15))

sns.kdeplot(df['Income'],ax=axes[0, 0],fill=True)
sns.kdeplot(df['Limit'],ax=axes[0, 1],fill=True)
sns.kdeplot(df['Rating'],ax=axes[0, 2],fill=True)
sns.kdeplot(df['Age'],ax=axes[1, 0],fill=True)
sns.kdeplot(df['Balance'],ax=axes[1, 1],fill=True)
sns.countplot(x=df['Cards'],ax=axes[1, 2])
sns.countplot(x=df['Education'],ax=axes[2, 0])
sns.countplot(x=df['Gender'],ax=axes[2, 1])
sns.countplot(x=df['Student'],ax=axes[2, 2])
sns.countplot(x=df['Married'],ax=axes[3, 0])
sns.countplot(x=df['Ethnicity'],ax=axes[3, 1])
```

Out[366]:

```
<AxesSubplot:xlabel='Ethnicity', ylabel='count'>
```



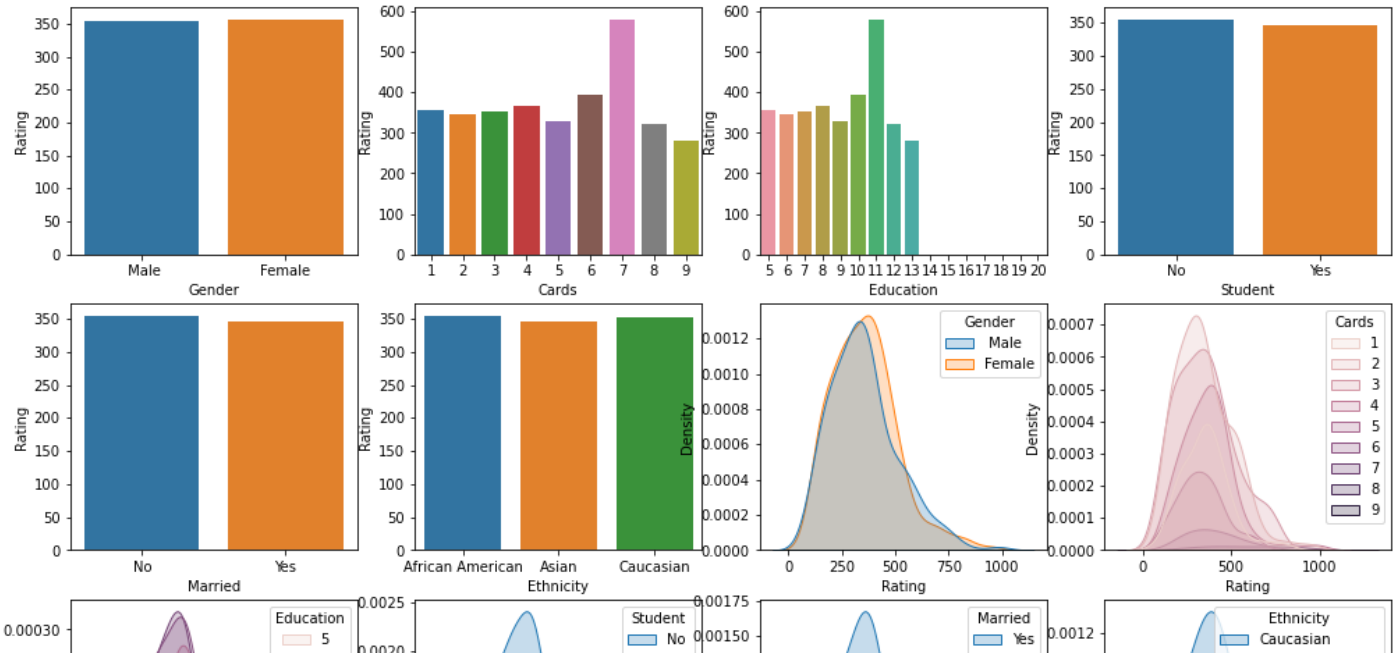
In [367]:

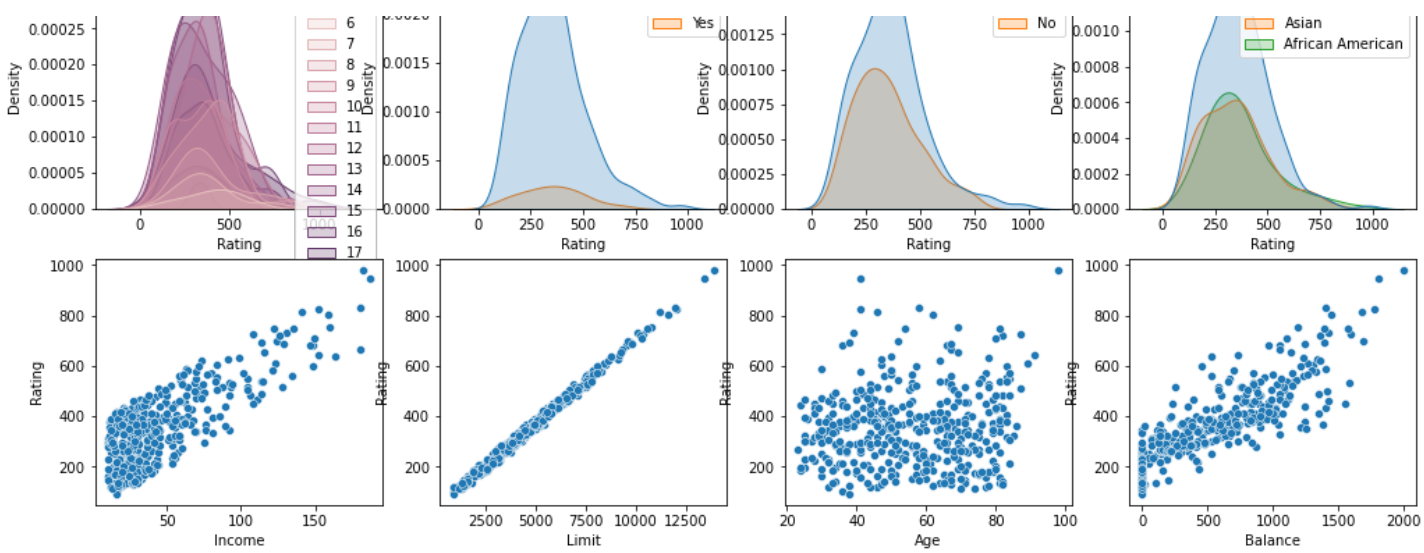
```
fig, axes = plt.subplots(4, 4, figsize=(17, 15))
df_Gender=df.groupby('Gender').mean('Rating')
df_Gender.reset_index(inplace=True)
df_Gender=df_Gender[['Rating','Gender']]
sns.barplot(x=df_Gender['Gender'],y=df_Gender['Rating'],ax=axes[0,0])
df_Cards=df.groupby('Cards').mean('Rating')
df_Cards.reset_index(inplace=True)
df_Cards=df_Cards[['Rating','Cards']]
sns.barplot(x=df_Cards['Cards'],y=df_Cards['Rating'],ax=axes[0,1])
df_Education=df.groupby('Education').mean('Rating')
df_Education.reset_index(inplace=True)
df_Education=df_Education[['Rating','Education']]
sns.barplot(x=df_Education['Education'],y=df_Cards['Rating'],ax=axes[0,2])
df_Student=df.groupby('Student').mean('Rating')
df_Student.reset_index(inplace=True)
df_Student=df_Student[['Rating','Student']]
sns.barplot(x=df_Student['Student'],y=df_Cards['Rating'],ax=axes[0,3])
df_Married=df.groupby('Married').mean('Rating')
df_Married.reset_index(inplace=True)
df_Married=df_Married[['Rating','Married']]
sns.barplot(x=df_Married['Married'],y=df_Cards['Rating'],ax=axes[1,0])
df_Ethnicity=df.groupby('Ethnicity').mean('Rating')
df_Ethnicity.reset_index(inplace=True)
df_Ethnicity=df_Ethnicity[['Rating','Ethnicity']]
sns.barplot(x=df_Ethnicity['Ethnicity'],y=df_Cards['Rating'],ax=axes[1,1])
sns.kdeplot(df['Rating'],fill=True,hue=df['Gender'],warn_singular=False,ax=axes[1,2])
sns.kdeplot(df['Rating'],fill=True,hue=df['Cards'],warn_singular=False,ax=axes[1,3])
sns.kdeplot(df['Rating'],fill=True,hue=df['Education'],warn_singular=False,ax=axes[2,0])
sns.kdeplot(df['Rating'],fill=True,hue=df['Student'],warn_singular=False,ax=axes[2,1])
sns.kdeplot(df['Rating'],fill=True,hue=df['Married'],warn_singular=False,ax=axes[2,2])
sns.kdeplot(df['Rating'],fill=True,hue=df['Ethnicity'],warn_singular=False,ax=axes[2,3])
sns.scatterplot(y=df['Rating'],x=df['Income'],ax=axes[3,0])
sns.scatterplot(y=df['Rating'],x=df['Limit'],ax=axes[3,1])
sns.scatterplot(y=df['Rating'],x=df['Age'],ax=axes[3,2])
sns.scatterplot(y=df['Rating'],x=df['Balance'],ax=axes[3,3])

'''
for ax in axes.flatten():
    plt.sca(ax)
    plt.xticks(rotation = 45)
'''
```

Out[367]:

```
<AxesSubplot:xlabel='Balance', ylabel='Rating'>
```



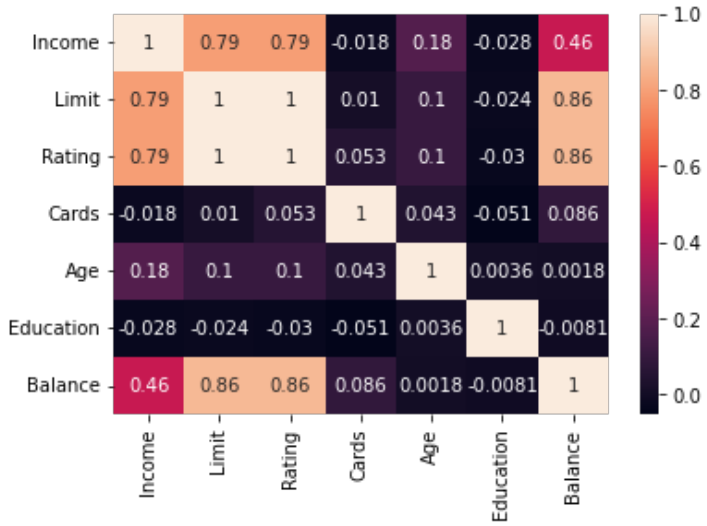


In [368]:

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

Out[368]:

<AxesSubplot:>

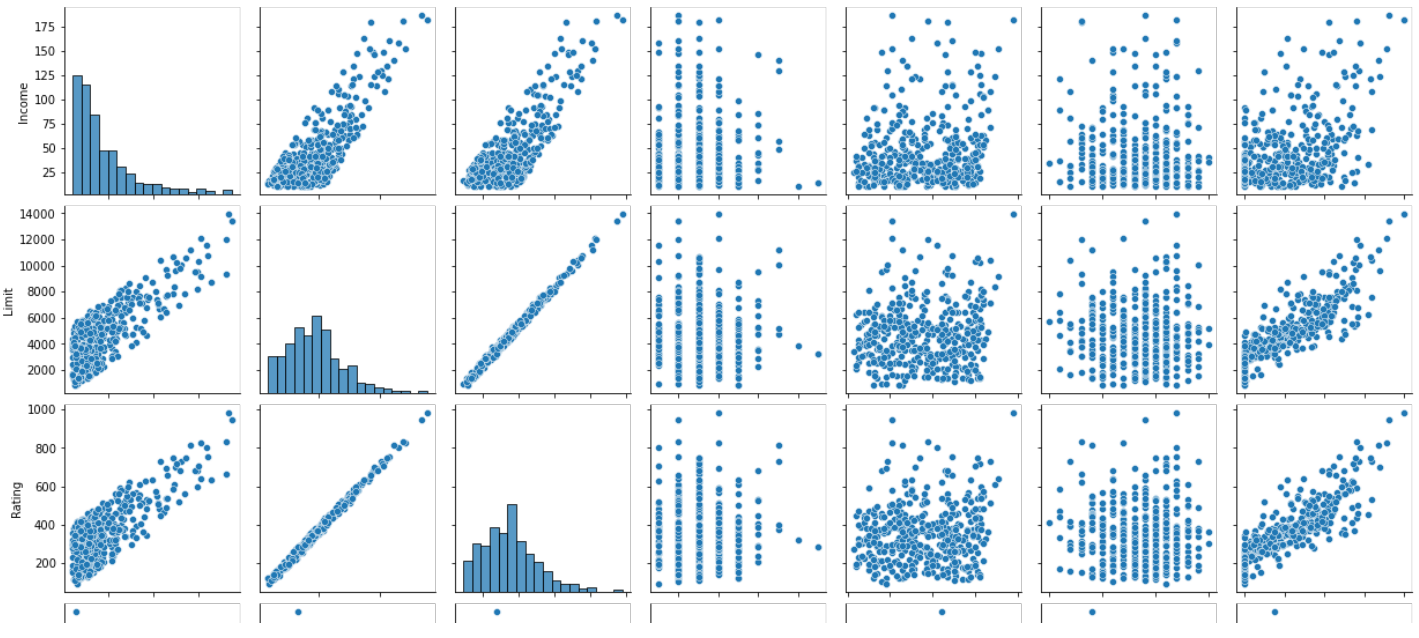


In [369]:

```
sns.pairplot(df)
```

Out[369]:

<seaborn.axisgrid.PairGrid at 0x27ef3c855b0>



Insights based on EDA

1.Income is having corelation with Limit and Balance

2.Income is right skewed, having Outliers beyond Upper Limit

3.People with 7 cards are having high mean Rating

4.Rating is having linear Trend with Income, Limit and Balance

2.Data Preprocessing

1.As we have seen earlier ,no null value is present (no treatment of null required)

2.As outliers are present but even target(Rating) is also following similar trend(no treatment of outliers required)

3.Converting Categorical text data into Categorical nominal type

In [371]:

```
categorical_text=['Gender','Student','Married','Ethnicity']
```

In [372]:

```
for i in categorical_text:
    df[i]=df[i].astype('category')
    df[i]=df[i].cat.codes
```

In [373]:

```
df.head()
```

Out[373]:

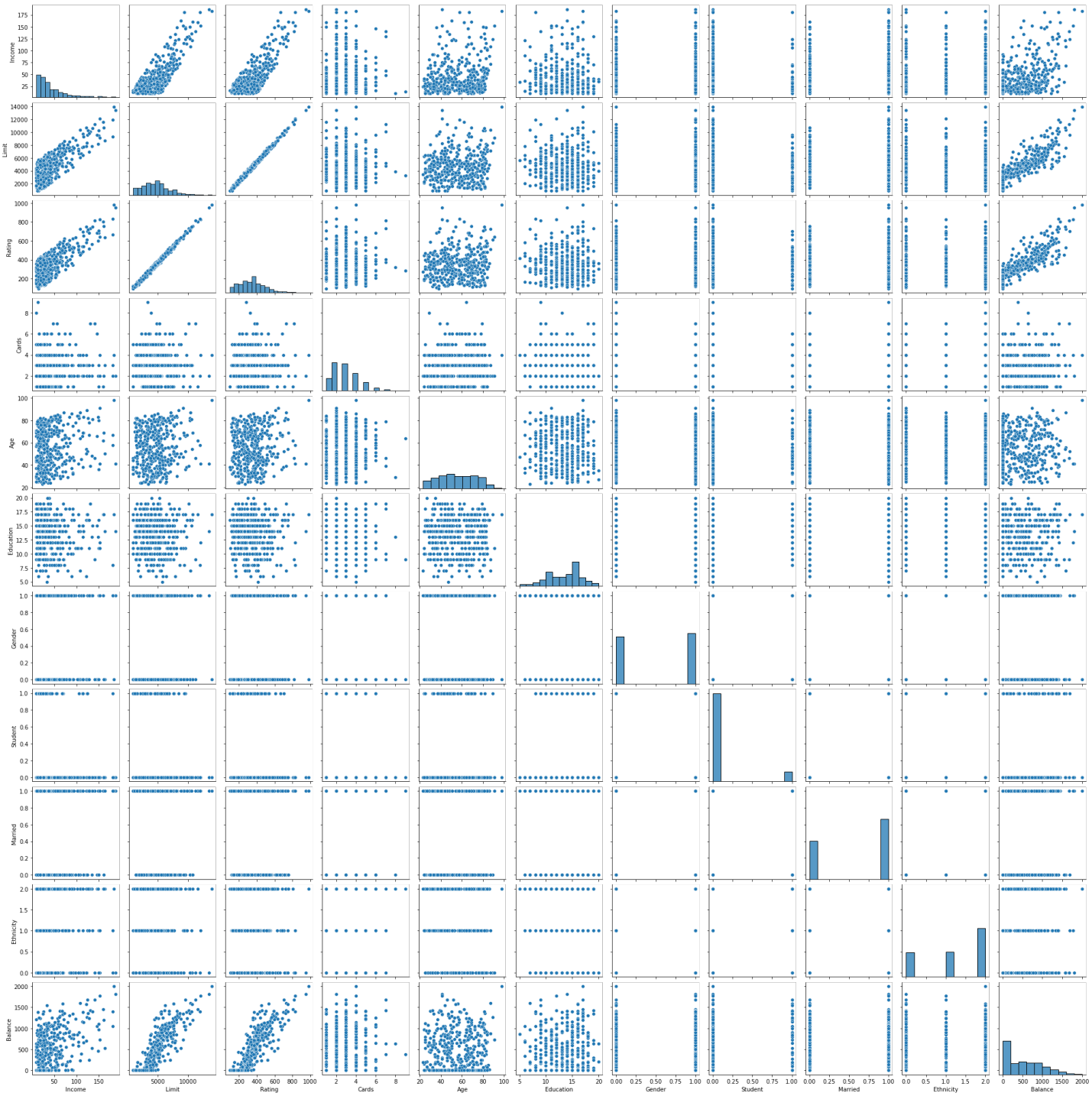
	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	0	0	1	2	333
1	106.025	6645	483	3	82	15	1	1	1	1	903
2	104.593	7075	514	4	71	11	0	0	0	1	580
3	148.924	9504	681	3	36	11	1	0	0	1	964
4	55.882	4897	357	2	68	16	0	0	1	2	331

In [374]:

```
sns.pairplot(df)
```

Out[374]:

```
<seaborn.axisgrid.PairGrid at 0x27f033c68b0>
```



In [375]:

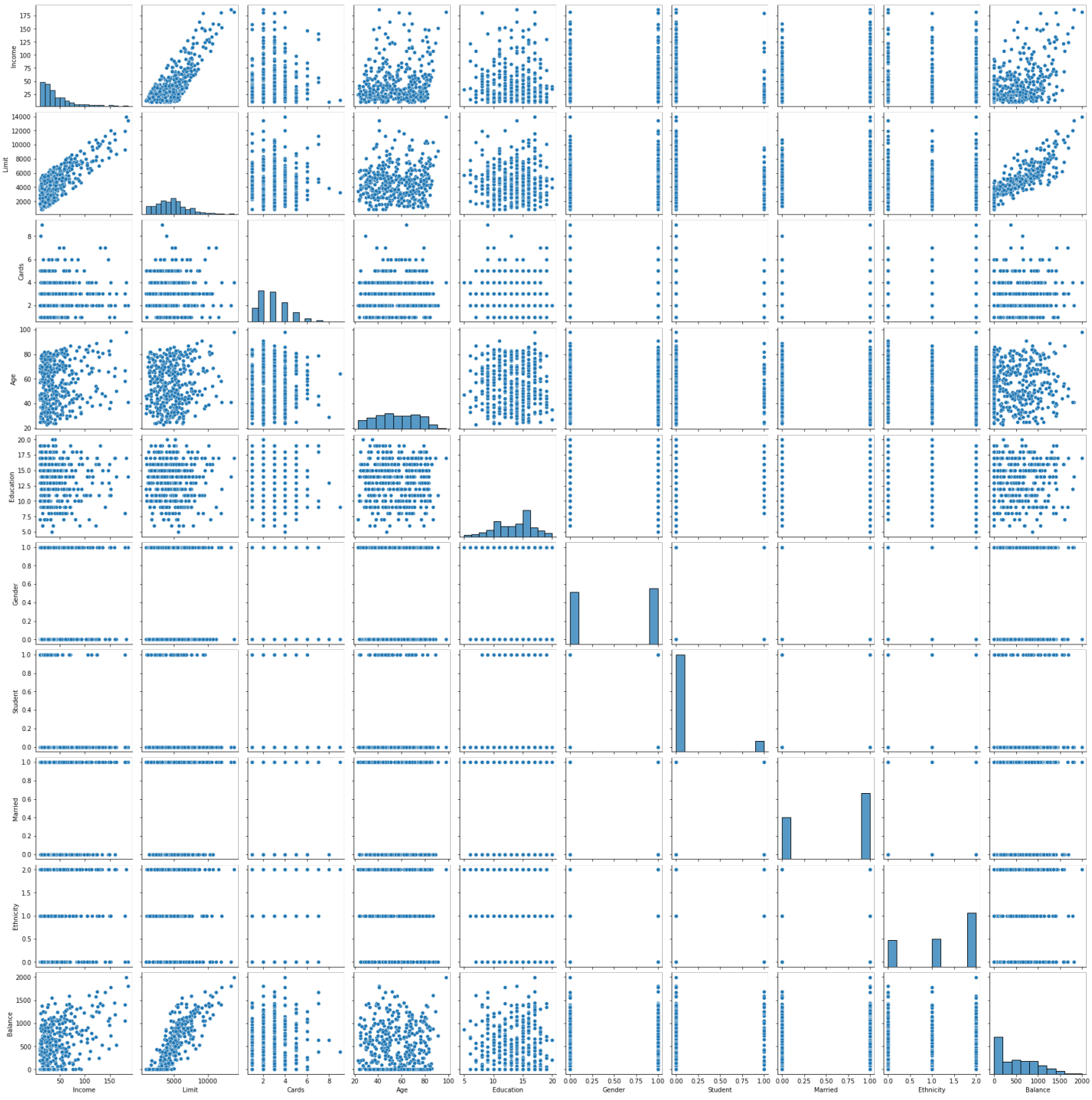
```
Y=df[['Rating']] # Target Variable
X=df[['Income','Limit','Cards','Age','Education','Gender','Student','Married','Ethnicity
','Balance']] # Predictors
```

In [376]:

```
sns.pairplot(X)
```

Out[376]:

```
<seaborn.axisgrid.PairGrid at 0x27f096ccd90>
```



In [377]:

```
Y.head()
```

Out[377]:

	Rating
0	**283**
1	**483**
2	**514**
3	**681**
4	**357**

In [378]:

```
X.head()
```

Out[378]:

	Income	Limit	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	2	34	11	0	0	1	2	333
1	106.025	6645	3	82	15	1	1	1	1	903
2	104.593	7075	4	71	11	0	0	0	1	580
3	148.924	9504	3	36	11	1	0	0	1	964
4	55.882	4897	2	68	16	0	0	1	2	331

3.Model building

In [379]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [380]:

```
from sklearn.model_selection import train_test_split
```

In [381]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

In [382]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

scaler = StandardScaler()
fit=scaler.fit(X_train)
X_train_Transformed = fit.transform(X_train)
```

In [383]:

```
X_train_Transformed
```

Out[383]:

```
array([[-0.34239421,  0.49599775,  0.74083028, ..., -1.2489996 ,
         0.90265989,  0.89259877],
       [-0.36169649, -0.75741407, -0.70911277, ...,  0.80064077,
        -1.47471596, -1.11312556],
       [-0.3407206 , -1.24695497, -0.70911277, ...,  0.80064077,
        -0.28602803, -1.11312556],
       ...,
       [ 1.28954049,  1.86934922, -1.4340843 , ...,  0.80064077,
        -0.28602803,  1.79149251],
       [-0.01353033, -0.12198915, -0.70911277, ...,  0.80064077,
         0.90265989,  0.61318361],
       [ 0.8707481 , -0.00885459,  0.74083028, ..., -1.2489996 ,
        -0.28602803, -0.87919659]])
```

In [384]:

```
Linear_reg = LinearRegression()
model=Linear_reg.fit(X_train_Transformed,y_train)
```

In [385]:

```
X_test_Transformed=fit.transform(X_test)
r_sq_train = model.score(X_train_Transformed, y_train)
```

```
print(f"R2 score with train data : {r_sq_train}")
r_sq_test = model.score(X_test_Transformed, y_test)
print(f"R2 score with test data : {r_sq_test}")
```

```
R2 score with train data : 0.9960896293099412
R2 score with test data : 0.99408286138675
```

In [386]:

```
arr=model.coef_
arr
```

Out[386]:

```
array([[ 4.47764102e+00,  1.49456051e+02,  6.24207308e+00,
         1.82964358e-01, -1.15714083e+00, -4.68403540e-01,
        -7.59977758e-01,  1.17947248e+00,  1.38198992e-01,
         4.37163809e+00]])
```

In [387]:

```
model.intercept_
```

Out[387]:

```
array([354.021875])
```

In [388]:

```
col=list(X.columns)
print("Predicted_Rating",end=' = ')
for i in range(len(col)):
    w=str(round(arr[0][i],2))
    x=col[i]
    print(f"{w}*{x}",end=' + ')
print(f"1x{model.intercept_[0]}")
```

```
Predicted_Rating = 4.48*Income + 149.46*Limit + 6.24*Cards + 0.18*Age + -1.16*Education +
-0.47*Gender + -0.76*Student + 1.18*Married + 0.14*Ethnicity + 4.37*Balance + 1x354.02187
5
```

Predicted_Rating = 4.48Income + 149.46Limit + 6.24Cards + 0.18Age + -1.16Education + -0.47Gender + -0.76Student + 1.18Married + 0.14Ethnicity + 4.37Balance + 1x354.021875

4.Testing the assumptions of linear regression model

4.1.Multicollinearity check by VIF score

In [389]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [390]:

```
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [391]:

```
calc_vif(pd.DataFrame(fit.transform(X),columns = col))
```

Out[391]:

	variables	VIF
0	Income	10.543934
1	Limit	41.890862
2	Cards	1.113955
3	Age	1.059916
4	Education	1.015075
5	Gender	1.008071
6	Student	2.751394
7	Married	1.022348
8	Ethnicity	1.008357
9	Balance	21.930672

AS Limit is having maximum VIF, so we need to again train model after droping Limit column

In [392]:

```
X.drop(columns=['Limit'],inplace=True)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
scaler = StandardScaler()
fit=scaler.fit(X_train)
X_train_Transformed = fit.transform(X_train)
Linear_reg = LinearRegression()
model=Linear_reg.fit(X_train_Transformed,y_train)
X_test_Transformed=fit.transform(X_test)
r_sq_train = model.score(X_train_Transformed, y_train)
print(f"R2 score with train data : {r_sq_train}")
r_sq_test = model.score(X_test_Transformed, y_test)
print(f"R2 score with test data : {r_sq_test}")
col=list(X.columns)
print("Predicted_Rating",end=' = ')
for i in range(len(col)):
    w=str(round(arr[0][i],2))
    x=col[i]
    print(f"({w})*{x}",end=' + ')
print(f"1x{round(model.intercept_[0],0)}")
```

```
R2 score with train data : 0.9748014157202345
R2 score with test data : 0.9723198439319113
Predicted_Rating = (4.48)*Income + (149.46)*Cards + (6.24)*Age + (0.18)*Education + (-1.1
6)*Gender + (-0.47)*Student + (-0.76)*Married + (1.18)*Ethnicity + (0.14)*Balance + 1x354
.0
```

```
C:\Users\rahul.kumar\AppData\Local\Temp\ipykernel_9660\1537406918.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  X.drop(columns=['Limit'],inplace=True)
```

In [393]:

```
calc_vif(pd.DataFrame(fit.transform(X),columns = col))
```

Out[393]:

	variables	VIF
0	Income	1.358214
1	Cards	1.021810
2	Age	1.052118

3	Education	1.013225
4	Gender	1.004549
5	Student	1.106659
6	Married	1.021816
7	Ethnicity	1.007388
8	Balance	1.416839

As VIF is less than 5 for each predictors, So we can freeze our model

R2 score with test data : 0.9723198439319113

Predicted_Rating = (4.48)xIncome + (149.46)xCards + (6.24)xAge + (0.18)xEducation + (-1.16)xGender + (-0.47)xStudent + (-0.76)xMarried + (1.18)xEthnicity + (0.14)xBalance + 1x354.0

****Note : Weights above mentioned is valid only if the columns are standardised***

4.2.Mean of residuals

In [394]:

```
y_predict = model.predict(fit.transform(X))
residual=y_predict-Y
residual.rename(columns = {'Rating':'Residual'}, inplace = True)
print(f"Mean Residul for the Model is : {round(residual.mean())} unit")
```

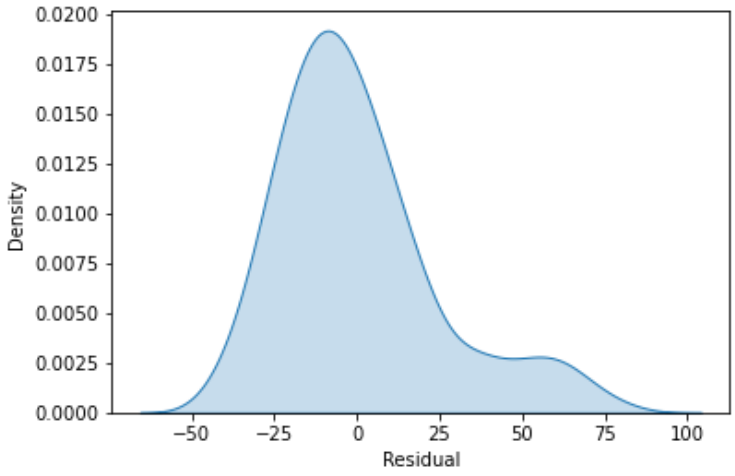
```
Mean Residul for the Model is : Residual    0.0
dtype: float64 unit
```

In [395]:

```
sns.kdeplot(residual['Residual'],fill=True)
```

Out[395]:

```
<AxesSubplot:xlabel='Residual', ylabel='Density'>
```



As the Mean of Residual is close to Zero, Hence Linear Regression model is correct

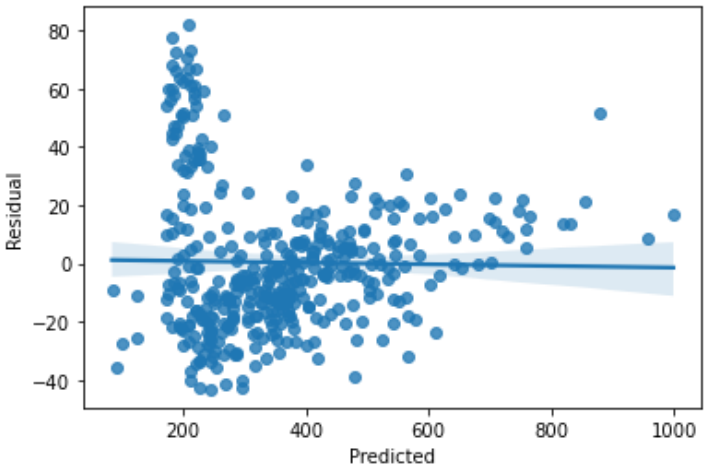
4.3.Linearity of variables

In [396]:

```
y_predict=pd.DataFrame(y_predict,columns = ['Predicted'])
```

In [411]:

```
ax=sns.regplot(x=y_predict['Predicted'],y=residual['Residual'],fit_reg=True)
```



As we can a see a linear trend. So, Linearity of variable is followed

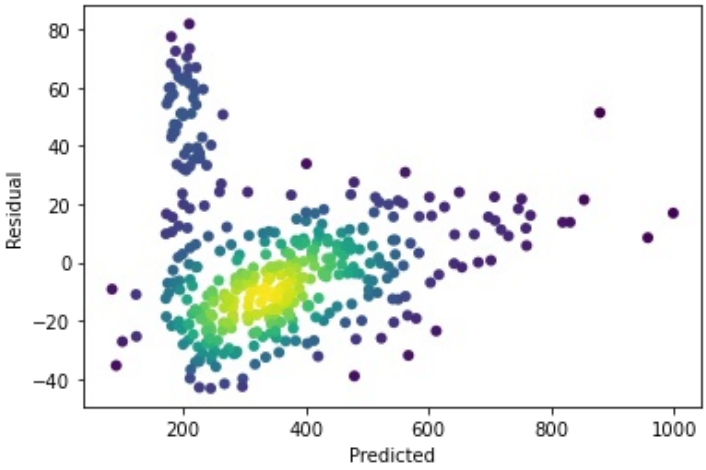
4.4.Test for Homoscedasticity

In [410]:

```
from scipy.stats import gaussian_kde
xy = np.vstack([y_predict['Predicted'],residual['Residual']])
z = gaussian_kde(xy)(xy)
sns.scatterplot(x=y_predict['Predicted'],y=residual['Residual'],c=z,edgecolor = 'none',ci=None)
```

Out[410]:

```
<AxesSubplot:xlabel='Predicted', ylabel='Residual'>
```

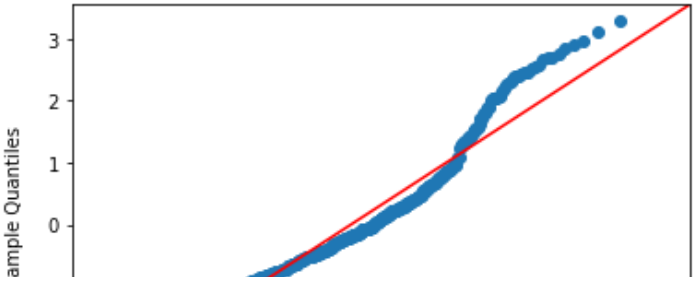


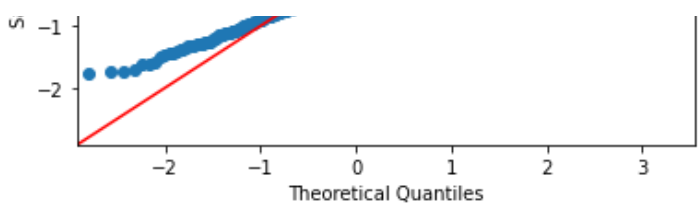
Homoscedasticity assumption is also adhered

4.5.Normality of residuals

In [399]:

```
QQplot=sm.qqplot(residual['Residual'], line ='45',fit=True)
```





QQ-Plot is justifying Normality of residuals

5.Model performance evaluation

5.1.Metrics checked - MAE, RMSE, R2, Adj R2

In [400]:

```
from statsmodels.tools.eval_measures import rmse
rmse = rmse(y_predict,Y)
print(f"RMSE for the Model is : {round(rmse[0],2)} unit")
```

```
RMSE for the Model is : 24.73 unit
```

In [401]:

```
from statsmodels.tools.eval_measures import meanabs
MAE=meanabs(y_predict,Y, axis=0)
print(f"MAE for the Model is : {round(MAE[0],2)} unit")
```

```
MAE for the Model is : 18.78 unit
```

In [404]:

```
r_sq_test = model.score(X_test_Transformed, y_test)
print(f"R2 score with test data : {round(r_sq_test,2)}")
```

```
R2 score with test data : 0.97
```

In [405]:

```
Adj_R = 1 - (1-r_sq_test)*(len(Y)-1)/(len(Y)-X.shape[1]-1)
print(f"Adjusted R2 score : {round(Adj_R,2)}")
```

```
Adjusted R2 score : 0.97
```

5.2.Train and test performances

In [406]:

```
r_sq_train = model.score(X_train_Transformed, y_train)
print(f"R2 score with train data : {round(r_sq_train,2)}")
r_sq_test = model.score(X_test_Transformed, y_test)
print(f"R2 score with test data : {round(r_sq_test,2)}")
```

```
R2 score with train data : 0.97
R2 score with test data : 0.97
```

5.3.Comments on the performance measures and if there is any need to improve the model or not

With R2 Score of 97% and RMSE of 24.73 credit score, our model is very good.

We can acheive some computational Performance improvement by reducing features like Balance, Student etc. with marginal decrease in accuracy as weights of these features are very less.

6.1 Comments on significance of predictor variables

6.1.Comments on significance of predictor variables

1.Cards, Age and Income is having Significant Impoact in Prediction

2.Balance, Student and Education is having least Impoact in Prediction, So we can ignore these to improve Computation Time

3.Income and Balance are having some colinearity, So we can also Remove income to get more confidence in weights

6.2.Comments on additional data sources for model improvement, model implementation in real world, potential business benefits from improving the model

1.For Implementation in real world, we can define a pipeline so that once it's triggered, all the steps shall run in order

2.We need to Check for Polynomial Regression if any scope for improvement is present

3.We need to Check for KNN Algorithm if we wnat to include all data

In [ ]: