In [91]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn import linear_model
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
%matplotlib inline
```

In [2]:
```
df=pd.read_csv("BankChurners.csv")
df
```

Out[2]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status
0	768805383	Existing Customer	45	M	3	High School	Married
1	818770008	Existing Customer	49	F	5	Graduate	Single
2	713982108	Existing Customer	51	M	3	Graduate	Married
3	769911858	Existing Customer	40	F	4	High School	Unknown
4	709106358	Existing Customer	40	M	3	Uneducated	Married
...	...	...	...	...	...	...	...
10122	772366833	Existing Customer	50	M	2	Graduate	Single
10123	710638233	Attrited Customer	41	M	2	Unknown	Divorced
10124	716506083	Attrited Customer	44	F	1	High School	Married
10125	717406983	Attrited Customer	30	M	2	Graduate	Unknown
10126	714337233	Attrited Customer	43	F	2	Graduate	Married

10127 rows × 23 columns

In [3]:
```
df.shape
```

Out[3]:
```
(10127, 23)
```

In [4]:
```
df.size
```

Out[4]:
```
232921
```

Loading [MathJax]/extensions/Safe.js

selecting the features we are going to use

In [5]:
```python
cols_to_use = ["Attrition_Flag","Customer_Age","Gender","Dependent_count","Education_Lev
df = df[cols_to_use]
df
```

Out[5]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Categ
0	Existing Customer	45	M	3	High School	Married	$60K - $8
1	Existing Customer	49	F	5	Graduate	Single	Less than $4
2	Existing Customer	51	M	3	Graduate	Married	$80K - $12
3	Existing Customer	40	F	4	High School	Unknown	Less than $4
4	Existing Customer	40	M	3	Uneducated	Married	$60K - $8
...	...	...	...	...	...	...	
10122	Existing Customer	50	M	2	Graduate	Single	$40K - $6
10123	Attrited Customer	41	M	2	Unknown	Divorced	$40K - $6
10124	Attrited Customer	44	F	1	High School	Married	Less than $4
10125	Attrited Customer	30	M	2	Graduate	Unknown	$40K - $6
10126	Attrited Customer	43	F	2	Graduate	Married	Less than $4

10127 rows × 9 columns

In [6]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Attrition_Flag   10127 non-null  object 
 1   Customer_Age     10127 non-null  int64  
 2   Gender           10127 non-null  object 
 3   Dependent_count  10127 non-null  int64  
 4   Education_Level  10127 non-null  object 
 5   Marital_Status   10127 non-null  object 
 6   Income_Category  10127 non-null  object 
 7   Card_Category    10127 non-null  object 
 8   Credit_Limit     10127 non-null  float64
dtypes: float64(1), int64(2), object(6)
memory usage: 712.2+ KB
```

In [7]:
```python
df.describe()
```

Out[7]:

	Customer_Age	Dependent_count	Credit_Limit
count	10127.000000	10127.000000	10127.000000
mean	46.325960	2.346203	8631.953698
std	8.016814	1.298908	9088.776650
min	26.000000	0.000000	1438.300000
25%	41.000000	1.000000	2555.000000
50%	46.000000	2.000000	4549.000000
75%	52.000000	3.000000	11067.500000
max	73.000000	5.000000	34516.000000

In [8]:
```
df[cols_to_use].nunique()
```

Out[8]:
```
Attrition_Flag        2
Customer_Age         45
Gender                2
Dependent_count       6
Education_Level       7
Marital_Status        4
Income_Category       6
Card_Category         4
Credit_Limit       6205
dtype: int64
```

looking for duplicate

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
30
```

treating the duplicate values

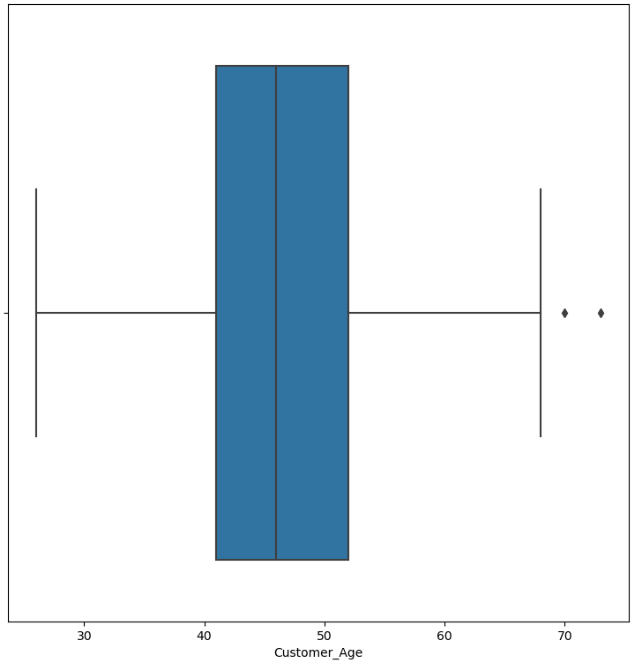
In [10]:
```
df.drop_duplicates(inplace=True)
```

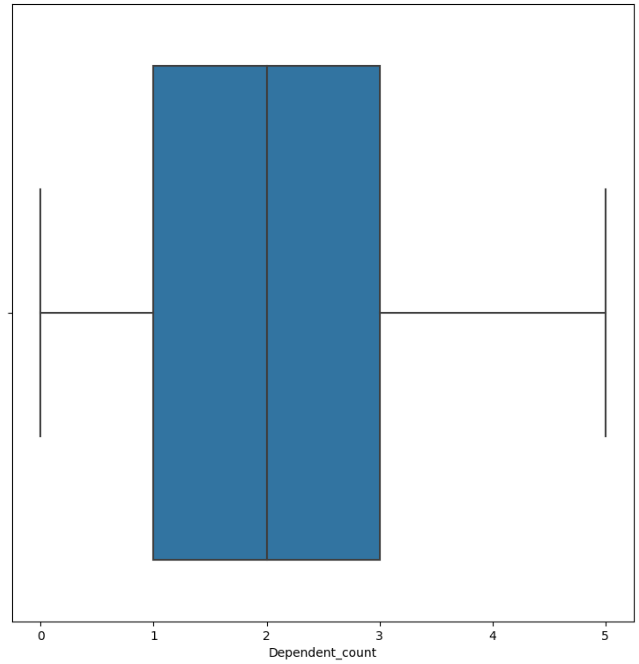
In [11]:
```
df.duplicated().sum()
```

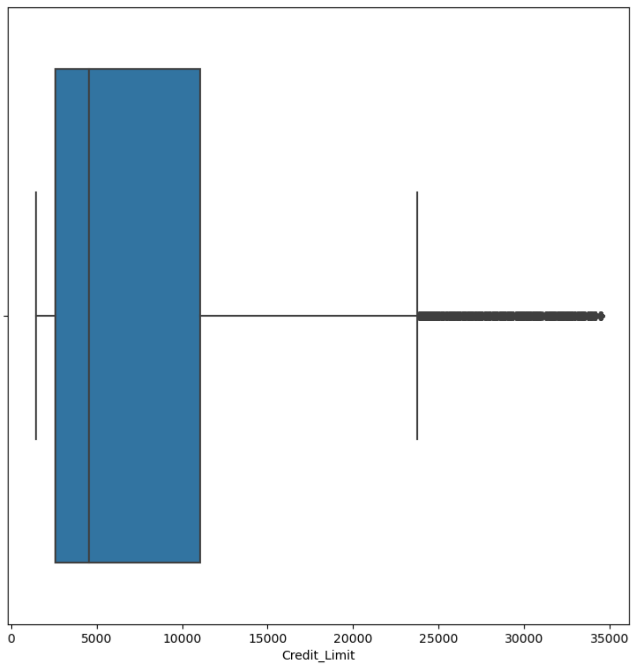
Out[11]:
```
0
```

looking for outliers

In [12]:
```
plt.figure(figsize=(20,20))
plt.subplot(2,2,1)
sns.boxplot(df["Customer_Age"])
plt.subplot(2,2,2)
sns.boxplot(df["Dependent_count"])
plt.subplot(2,2,3)
sns.boxplot(df["Credit_Limit"])
plt.show()
```







insights:

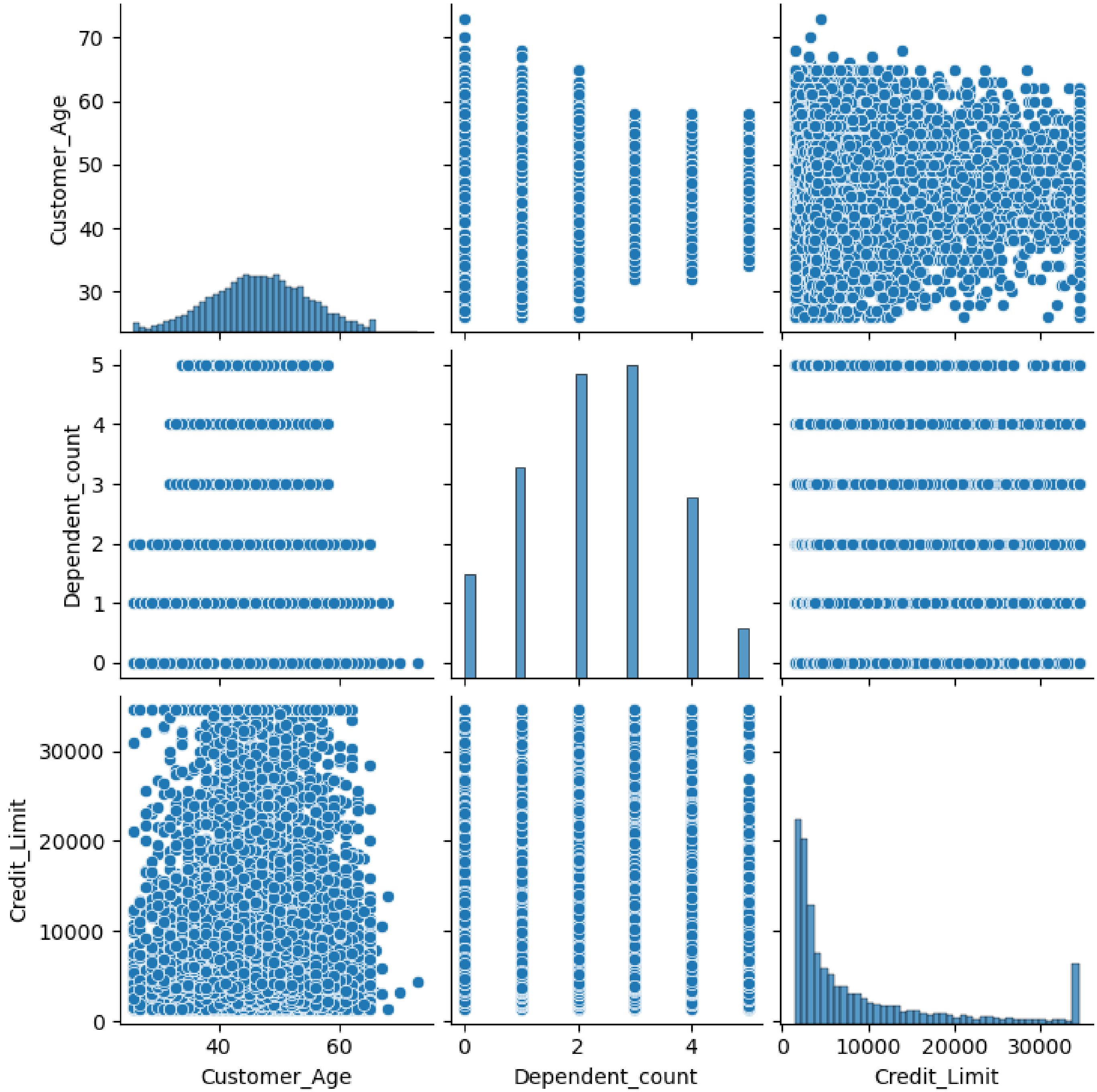
1. As we can see the credit_limit have a lot of outliers but can't remove them because they are considered to considrable outliers

data visualization

In [13]:
```
sns.pairplot(df)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x1ea70016df0>
```

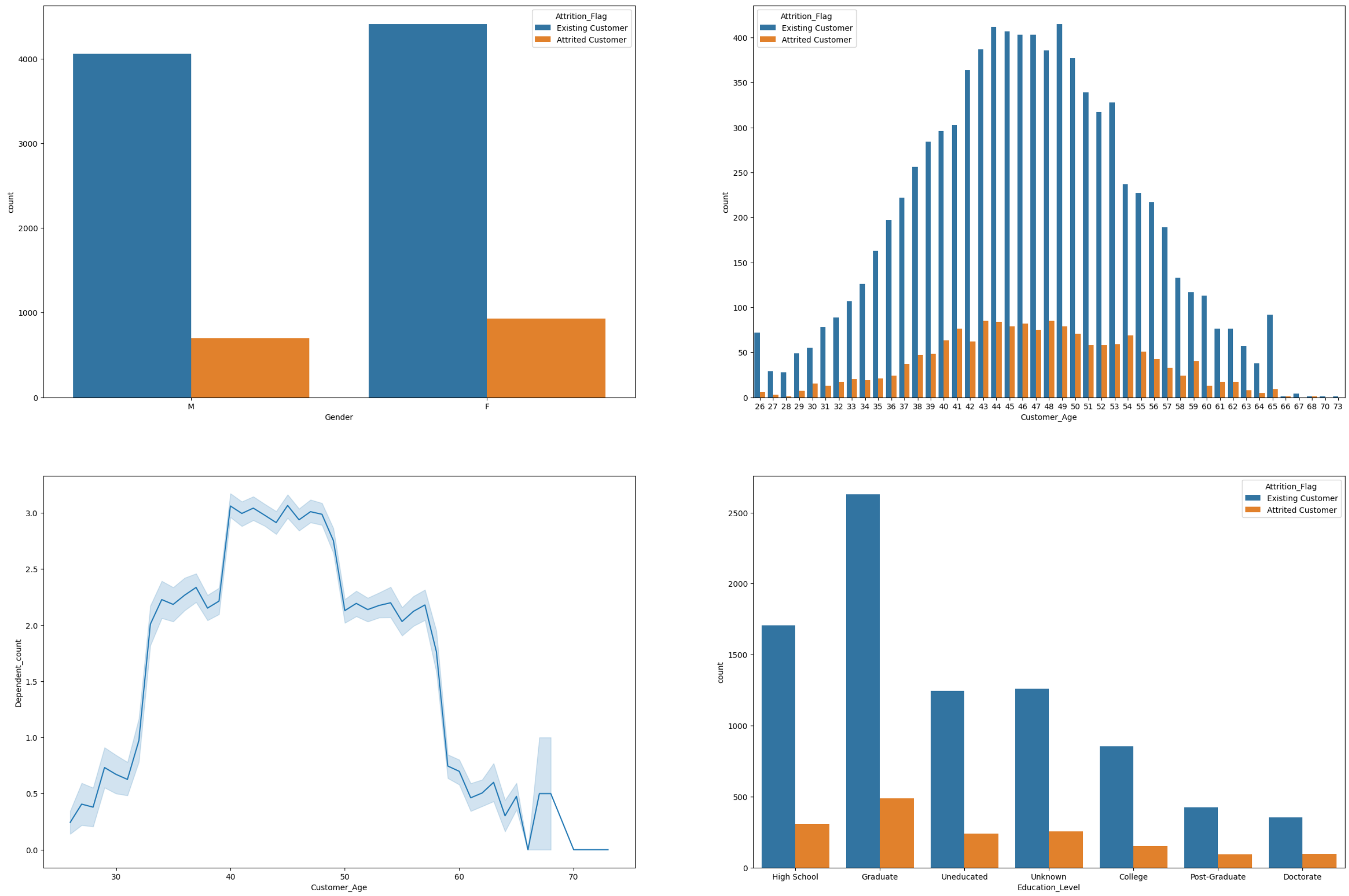
Loading [MathJax]/extensions/Safe.js



In [14]:

```
plt.figure(figsize=(30,20))
plt.subplot(2,2,1)
sns.countplot("Gender",hue="Attrition_Flag",data=df)
plt.subplot(2,2,2)
sns.countplot("Customer_Age",hue="Attrition_Flag",data=df)
plt.subplot(2,2,3)
sns.lineplot("Customer_Age","Dependent_count",data=df)
plt.subplot(2,2,4)
sns.countplot("Education_Level",hue="Attrition_Flag",data=df)
```

Out[14]: <AxesSubplot:xlabel='Education_Level', ylabel='count'>



insight:

1. As we can see customer who are doing business with the company are camparitively higher than the customer who has stoped doing the business with the company

   In both the casses no. of females as higher

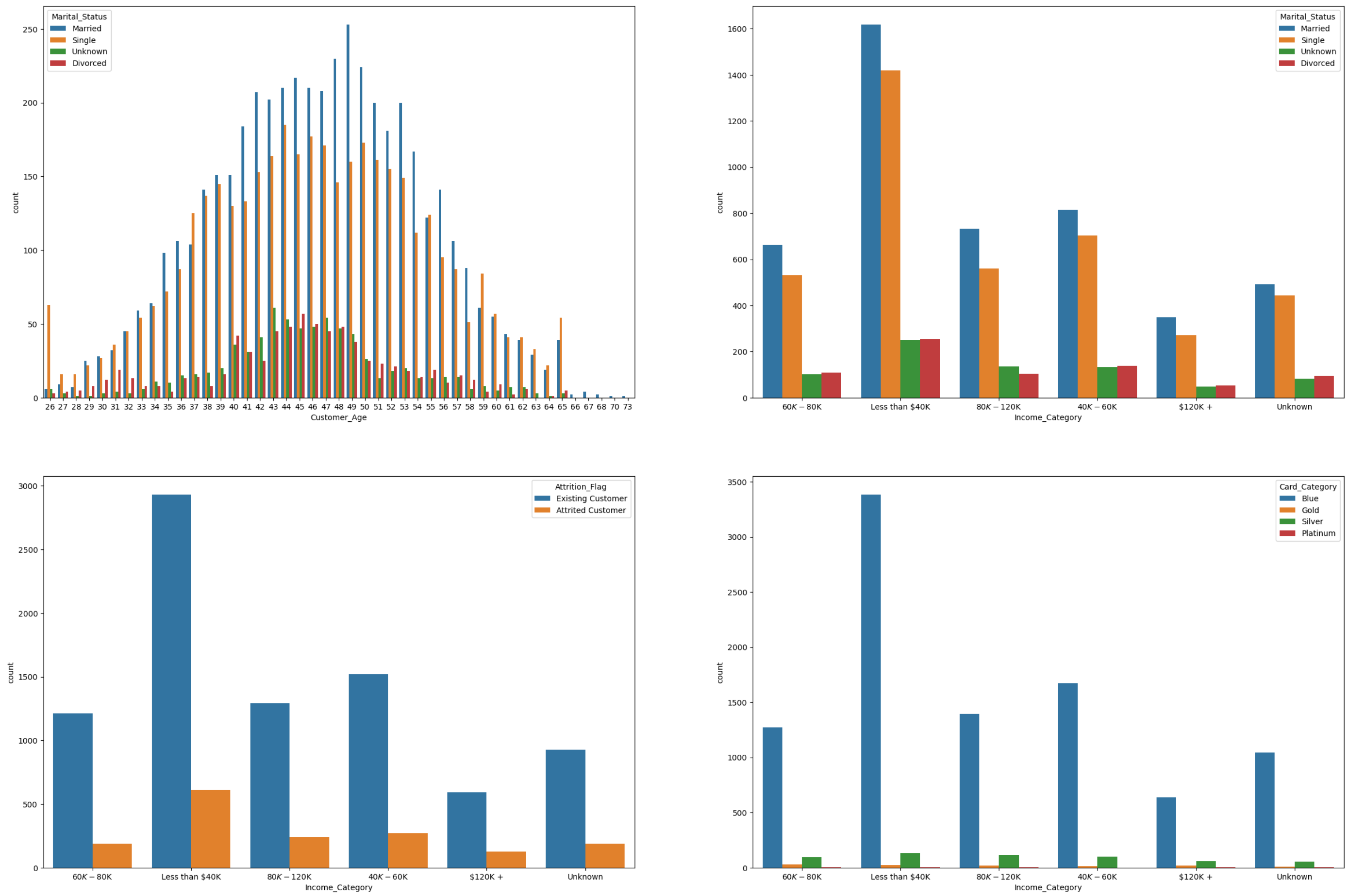
1. As we can see that from the age of 29 csutomer starts leaving the card company maybe they are getting good offers from other other companies

1. As we can see that as the age increases the dependent count increases also there is a drastic increase in dependent count after 30 (as we know most of the people in this age group get married or have kids) this is aloso considered while giving loan because higher the dependent count higher the financial responsiblity is and there are low chances that coustomer will pay back the loan

2. As we can see that the the people who have high school education and are graduate are higher as compared to other educational levels

In [15]:
```
plt.figure(figsize=(30,20))
plt.subplot(2,2,1)
sns.countplot("Customer_Age",hue="Marital_Status",data=df)
plt.subplot(2,2,2)
sns.countplot("Income_Category",hue="Marital_Status",data=df)
plt.subplot(2,2,3)
sns.countplot("Income_Category",hue="Attrition_Flag",data=df)
plt.subplot(2,2,4)
sns.countplot("Income_Category",hue="Card_Category",data=df)
```

Out[15]: <AxesSubplot:xlabel='Income_Category', ylabel='count'>



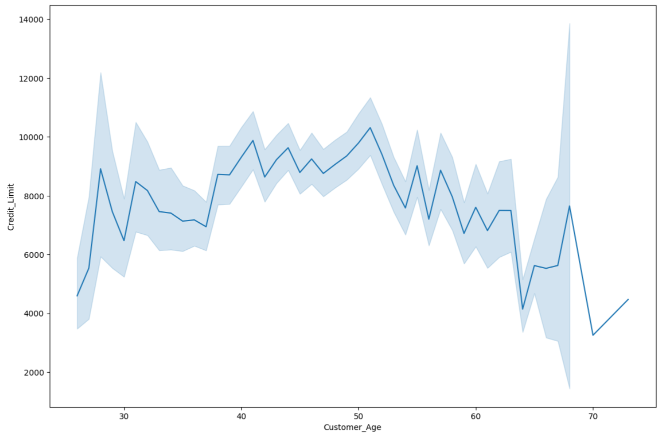
insight:

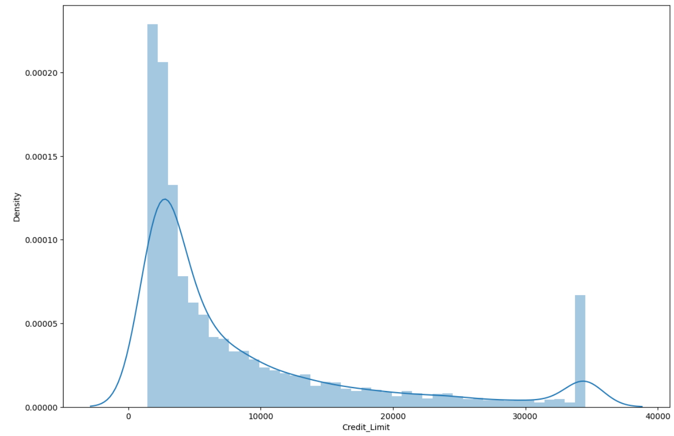
1. As we saw that there was a drastic increase in dependent count after the age 33 also here we can see that there is a increase in married people after the age of 32
2. We can see that most of the customer lies between the income catagory less than 40k also we saw that in this catagory married people are higher so we can consider this income catagory to be highly risky to give loans.

In [16]:

```
plt.figure(figsize=(30,20))
plt.subplot(2,2,1)
sns.lineplot("Customer_Age","Credit_Limit",data=df)
plt.subplot(2,2,2)
sns.distplot(df["Credit_Limit"])
```

Out[16]: <AxesSubplot:xlabel='Credit_Limit', ylabel='Density'>





as we can see there are multipe categorical columns. for this we use label and onehot encoding

In [17]:
```
le=LabelEncoder()
df["Attrition_Flag"]=le.fit_transform(df["Attrition_Flag"])
df["Gender"]=le.fit_transform(df["Gender"])
```

In [18]:
```
ohe=pd.get_dummies(df,columns=["Education_Level","Marital_Status","Income_Category","Car
ohe
```

Out[18]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Credit_Limit	Education_Level_College	Educati
0	1	45	1	3	12691.0	0	
1	1	49	0	5	8256.0	0	
2	1	51	1	3	3418.0	0	
3	1	40	0	4	3313.0	0	
4	1	40	1	3	4716.0	0	
...	...	...	...	...	...	...	
10122	1	50	1	2	4003.0	0	
10123	0	41	1	2	4277.0	0	
10124	0	44	0	1	5409.0	0	
10125	0	30	1	2	5281.0	0	
10126	0	43	0	2	10388.0	0	

10097 rows × 26 columns

In [19]:
```
df=ohe
```

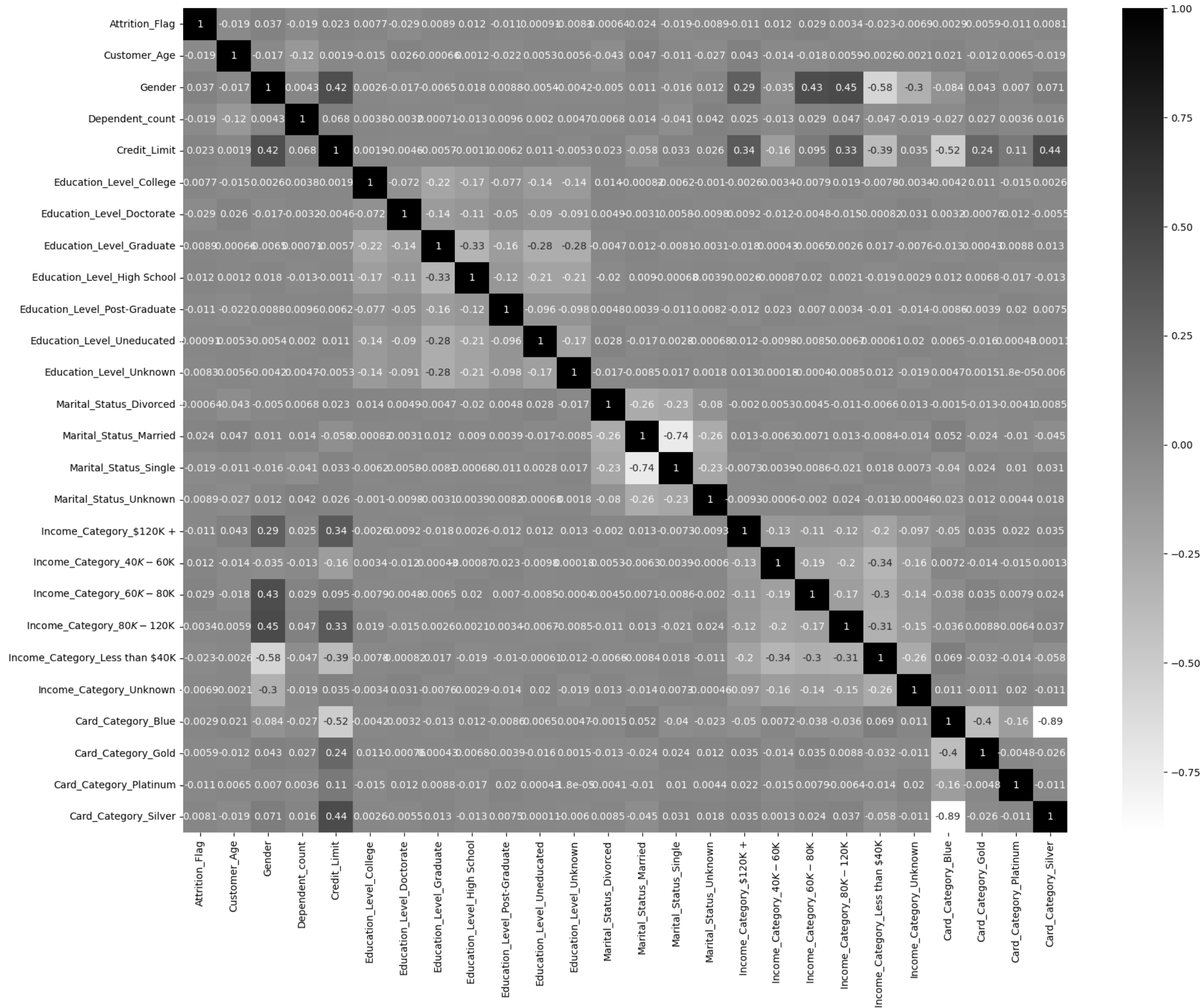
looking for correlation

```
gsize=(20,15))
```

```
sns.heatmap(ohe.corr(),annot=True,cmap="gray_r")
```

Out[20]:

```
<AxesSubplot:>
```



train test split

In [21]:

```
x=df.drop("Attrition_Flag",axis=1)
y=df["Attrition_Flag"]
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.700,random_state=100)
```

In [22]:

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [23]:

```
x_train.shape
```

Out[23]:

```
(7067, 25)
```

In [24]:

```
x_test.shape
```

Out[24]:

```
(3030, 25)
```

model building

1. Decision tree

In [25]:
```
dtc = DecisionTreeClassifier(criterion='gini', max_depth=7, random_state=0)
dtc.fit(x_train, y_train)
```
Out[25]:
```
DecisionTreeClassifier(max_depth=7, random_state=0)
```
In [26]:
```
y_pred = dtc.predict(x_test)
```
In [27]:
```
accuracy_score(y_test,y_pred)
```
Out[27]:
```
0.8366336633663366
```

using grid search to improve performance

In [28]:
```
dtc.get_params()
```
Out[28]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 7,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 0,
 'splitter': 'best'}
```
In [29]:
```
params={"max_depth":[2,5,20,30],"max_leaf_nodes":[5,10,20,50,100,250,270],"min_samples_s
dtct=GridSearchCV(dtc,params,cv=5)
dtct.fit(x_train,y_train)
```
Out[29]:
```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(max_depth=7, random_state=0),
             param_grid={'max_depth': [2, 5, 20, 30],
                         'max_leaf_nodes': [5, 10, 20, 50, 100, 250, 270],
                         'min_samples_split': [1, 2, 5, 8],
                         'random_state': [20, 50, 100]})
```
In [30]:
```
dtct.best_params_
```
Out[30]:
```
{'max_depth': 2,
 'max_leaf_nodes': 5,
 'min_samples_split': 2,
 'random_state': 20}
```
In [31]:
```
dtct.best_score_
```
Out[31]:
```
0.8369889218221186
```
In [32]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.30      0.03      0.06       473
           1       0.85      0.99      0.91      2557

    accuracy                           0.84      3030
   macro avg       0.57      0.51      0.49      3030
weighted avg       0.76      0.84      0.78      3030
```

2. KNN

In [33]:
```
knn=KNeighborsClassifier()
```

In [34]:
```
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
```

In [35]:
```
accuracy_score(y_test,y_pred)
```

Out[35]:
```
0.8181518151815181
```

using grid search to improve performance

In [36]:
```
knn.get_params()
```

Out[36]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [37]:
```
knn_params={"n_neighbors":[10,20,30,40,100],'leaf_size':[20,30,60,70,80]}
knn_cv_model = GridSearchCV(knn,knn_params, cv=20)
knn_cv=knn_cv_model.fit(x_train,y_train)
```

In [38]:
```
knn_cv.best_params_
```

Out[38]:
```
{'leaf_size': 20, 'n_neighbors': 30}
```

In [39]:
```
accuracy=knn_cv.best_score_
accuracy
```

Out[39]:
```
0.8369896448520351
```

3. svm

In [40]:
```
svm=SVC(kernel="linear")
```

In [41]:
```
svm.fit(x_train,y_train)
```

Out[41]:
```
SVC(kernel='linear')
```

In [42]:
```
y_pred=svm.predict(x_test)
```

In [43]:
```
accuracy_score(y_test,y_pred)
```
Out[43]:
```
0.8438943894389439
```

using grid search to improve performance

In [44]:
```
svm.get_params()
```
Out[44]:
```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'linear',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

In [45]:
```
svm_param={"cache_size":[10,20,30,40],"C":[2.0,3.0,4.0,5.0],"degree":[1,2,3,4,5,6,7,8]}
svm_tuning = GridSearchCV(svm,svm_param, cv=7)
svm_tuning=svm_tuning.fit(x_train,y_train)
```

In [46]:
```
svm_tuning.best_params_
```
Out[46]:
```
{'C': 2.0, 'cache_size': 10, 'degree': 1}
```

In [47]:
```
svm_tuning.best_score_
```
Out[47]:
```
0.836989022419161
```

In [ ]:

4. random forest

In [48]:
```
rfc=RandomForestClassifier(criterion='entropy')
```

In [49]:
```
rfc.fit(x_train,y_train)
```
Out[49]:
```
RandomForestClassifier(criterion='entropy')
```

In [50]:
```
y_pred_rfc=rfc.predict(x_test)
```

In [51]:
```
accuracy_score(y_test,y_pred_rfc)
```
Out[51]:
```
0.8224422442244225
```

In [52]:
```
rfc.get_params()
```

Out[52]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [84]:
```
params={"max_depth":[2,5,10,30],"min_samples_split":[5,10,30],"min_impurity_decrease":[1
rfct=GridSearchCV(rfc,params,cv=5)
rfct.fit(x_train,y_train)
```

Out[84]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(criterion='entropy'),
             param_grid={'max_depth': [2, 5, 10, 30],
                         'min_impurity_decrease': [1.0, 2.0],
                         'min_samples_split': [5, 10, 30],
                         'n_estimators': [20, 40], 'random_state': [34, 40]})
```

In [85]:
```
rfct.best_score_
```

Out[85]:
```
0.8369889218221186
```

In [86]:
```
rfct.best_params_
```

Out[86]:
```
{'max_depth': 2,
 'min_impurity_decrease': 1.0,
 'min_samples_split': 5,
 'n_estimators': 20,
 'random_state': 34}
```

5. logestic regression

In [76]:
```
lreg=LogisticRegression()
```

In [77]:
```
lreg.fit(x_train,y_train)
```

Out[77]:
```
LogisticRegression()
```

In [78]:
```
y_pred=lreg.predict(x_test)
```

In [79]:
```
accuracy_score(y_test,y_pred)
```

Out[79]:
```
0.8438943894389439
```

In [80]:
```
lreg.get_params()
```

Out[80]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [87]:
```
params={"max_iter":[20,100],"C":[0.05,1.0]}
logetun=GridSearchCV(lreg,params,cv=20)
logetun.fit(x_train,y_train)
```

Out[87]:
```
GridSearchCV(cv=20, estimator=LogisticRegression(),
             param_grid={'C': [0.05, 1.0], 'max_iter': [20, 100]})
```

In [88]:
```
logetun.best_params_
```

Out[88]:
```
{'C': 0.05, 'max_iter': 20}
```

In [89]:
```
logetun.best_score_
```

Out[89]:
```
0.8369896448520351
```

6. Adaboost

In [92]:
```
ada=AdaBoostClassifier()
ada.fit(x_train,y_train)
```

Out[92]:
```
AdaBoostClassifier()
```

In [94]:
```
accuracy_score(y_test,y_pred)
```

Out[94]:
```
0.8438943894389439
```

using gridsearchcv() to improve model performance

In [95]:
```
ada.get_params()
```

Out[95]:
```
{'algorithm': 'SAMME.R',
 'base_estimator': None,
 'learning_rate': 1.0,
 'n_estimators': 50,
 'random_state': None}
```

In [119…
```
params={"learning_rate":[0.002,0.05,0.2,0.5,1.0,2,0,3.0],"n_estimators":[200,250,270],"r
adat=GridSearchCV(ada,params,cv=7)
adat.fit(x_train,y_train)
```

Out[119]:
```
GridSearchCV(cv=7, estimator=AdaBoostClassifier(),
             param_grid={'learning_rate': [0.002, 0.05, 0.2, 0.5, 1.0, 2, 0,
                                           3.0],
                         'n_estimators': [200, 250, 270],
                         'random_state': [5, 100]})
```

In [116…:
```
adat.best_params_
```

Out[116]:
```
{'learning_rate': 0.05, 'n_estimators': 200, 'random_state': 5}
```

In [117…:
```
adat.best_score_
```

Out[117]:
```
0.8369888729059065
```

accuracy of all the model

logestic regression = 0.8438

Decision tree = 0.8369

random forest = 0.8369

adaboost = 0.8469

knn = 0.8369

svm = 0.8438

as we can see we got the highest accuracy for adaboost classifier

In [ ]:
```

```