In [103]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [104]:
```
data = pd.read_csv('train data credit card.csv')
data
```

Out[104]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	Female	73	RG268	Other	X3	43	No	1045696	No	0
1	IDD62UNG	Female	30	RG277	Salaried	X1	32	No	581988	No	0
2	HD3DSEMC	Female	56	RG268	Self_Employed	X3	26	No	1484315	Yes	0
3	BF3NC7KV	Male	34	RG270	Salaried	X1	19	No	470454	No	0
4	TEASRWXV	Female	30	RG282	Salaried	X1	33	No	886787	No	0
...	...	...	...	...	...	...	...	...	...	...	...
245720	BPAWWXZN	Male	51	RG284	Self_Employed	X3	109	NaN	1925586	No	0
245721	HFNB7JY8	Male	27	RG268	Salaried	X1	15	No	862952	Yes	0
245722	GEHAUCWT	Female	26	RG281	Salaried	X1	13	No	670659	No	0
245723	GE7V8SAH	Female	28	RG273	Salaried	X1	31	No	407504	No	0
245724	BOCZSWLJ	Male	29	RG269	Salaried	X1	21	No	1129276	No	0

245725 rows × 11 columns

In [105]:
```
data.shape
```

Out[105]:
```
(245725, 11)
```

In [106]:
```
data.describe()
```

Out[106]:

	Age	Vintage	Avg_Account_Balance	Is_Lead
count	245725.000000	245725.000000	2.457250e+05	245725.000000
mean	43.856307	46.959141	1.128403e+06	0.237208
std	14.828672	32.353136	8.529364e+05	0.425372
min	23.000000	7.000000	2.079000e+04	0.000000
25%	30.000000	20.000000	6.043100e+05	0.000000
50%	43.000000	32.000000	8.946010e+05	0.000000
75%	54.000000	73.000000	1.366666e+06	0.000000
max	85.000000	135.000000	1.035201e+07	1.000000

In [107]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 245725 entries, 0 to 245724
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ID                   245725 non-null  object
 1   Gender               245725 non-null  object
 2   Age                  245725 non-null  int64 
 3   Region_Code          245725 non-null  object
 4   Occupation           245725 non-null  object
 5   Channel_Code         245725 non-null  object
 6   Vintage              245725 non-null  int64 
 7   Credit_Product       216400 non-null  object
 8   Avg_Account_Balance  245725 non-null  int64 
 9   Is_Active            245725 non-null  object
 10  Is_Lead              245725 non-null  int64 
dtypes: int64(4), object(7)
memory usage: 20.6+ MB
```

In [108]:
```
data.head()
```

Out[108]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	Female	73	RG268	Other	X3	43	No	1045696	No	0
1	IDD62UNG	Female	30	RG277	Salaried	X1	32	No	581988	No	0
2	HD3DSEMC	Female	56	RG268	Self_Employed	X3	26	No	1484315	Yes	0
3	BF3NC7KV	Male	34	RG270	Salaried	X1	19	No	470454	No	0
4	TEASRWXV	Female	30	RG282	Salaried	X1	33	No	886787	No	0

In [109]:
```
data.tail()
```

Out[109]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
245720	BPAWWXZN	Male	51	RG284	Self_Employed	X3	109	NaN	1925586	No	0
245721	HFNB7JY8	Male	27	RG268	Salaried	X1	15	No	862952	Yes	0
245722	GEHAUCWT	Female	26	RG281	Salaried	X1	13	No	670659	No	0
245723	GE7V8SAH	Female	28	RG273	Salaried	X1	31	No	407504	No	0
245724	BOCZSWLJ	Male	29	RG269	Salaried	X1	21	No	1129276	No	0

In [110]:
```
data.isnull().sum()
```

Out[110]:
```
ID                        0
Gender                    0
Age                       0
Region_Code               0
Occupation                0
Channel_Code              0
Vintage                   0
Credit_Product        29325
Avg_Account_Balance       0
Is_Active                 0
Is_Lead                   0
dtype: int64
```

In [111]:
```
data.isnull().sum()/data.shape[0]*100
```

Out[111]:
```
ID                     0.000000
Gender                 0.000000
Age                    0.000000
Region_Code            0.000000
Occupation             0.000000
Channel_Code           0.000000
Vintage                0.000000
Credit_Product        11.934073
Avg_Account_Balance    0.000000
Is_Active              0.000000
Is_Lead                0.000000
dtype: float64
```

In [112]:
```
data.Credit_Product.unique()
```

Out[112]:
```
array(['No', nan, 'Yes'], dtype=object)
```

In [113]:
```
data['Credit_Product'] = data['Credit_Product'].fillna('NA')
```

In [114]:
```
data.Credit_Product.unique()
```

Out[114]:
```
array(['No', 'NA', 'Yes'], dtype=object)
```

In [115]:
```
data.isnull().sum()
```

Out[115]:
```
ID                    0
Gender                0
Age                   0
Region_Code           0
Occupation            0
Channel_Code          0
Vintage               0
Credit_Product        0
Avg_Account_Balance   0
Is_Active             0
Is_Lead               0
dtype: int64
```

In [116]:
```
data['Is_Lead'].value_counts()
```

Out[116]:
```
0    187437
1     58288
Name: Is_Lead, dtype: int64
```

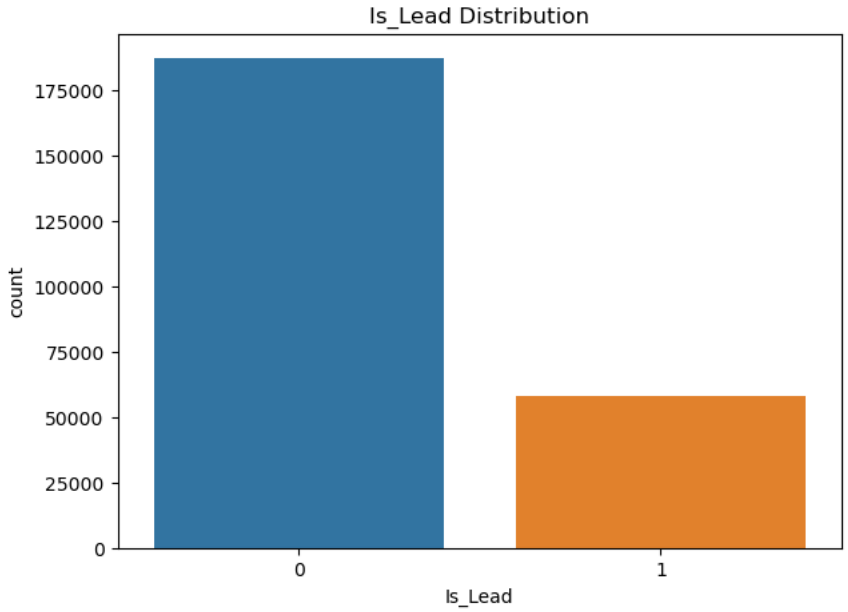
In [117]:
```
data.head()
```

Out[117]:

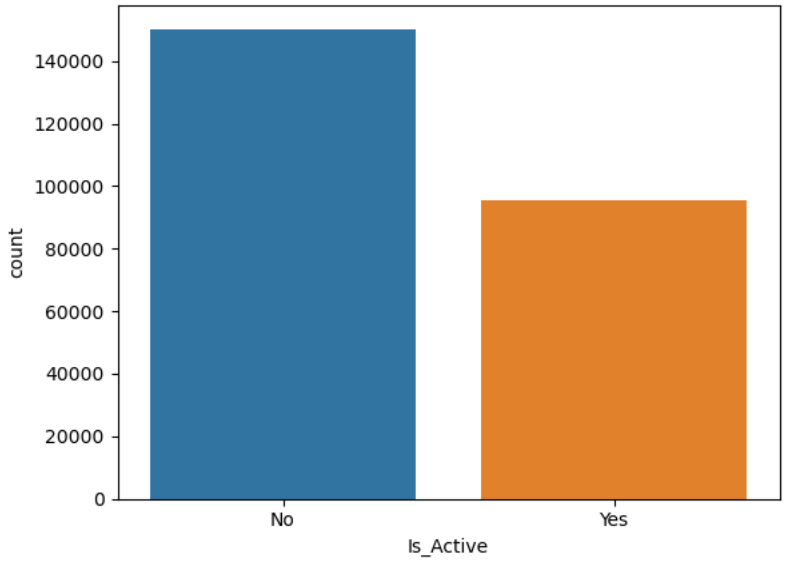
	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	Female	73	RG268	Other	X3	43	No	1045696	No	0
1	IDD62UNG	Female	30	RG277	Salaried	X1	32	No	581988	No	0
2	HD3DSEMC	Female	56	RG268	Self_Employed	X3	26	No	1484315	Yes	0
3	BF3NC7KV	Male	34	RG270	Salaried	X1	19	No	470454	No	0
4	TEASRWXV	Female	30	RG282	Salaried	X1	33	No	886787	No	0

VISUALIZATIONS

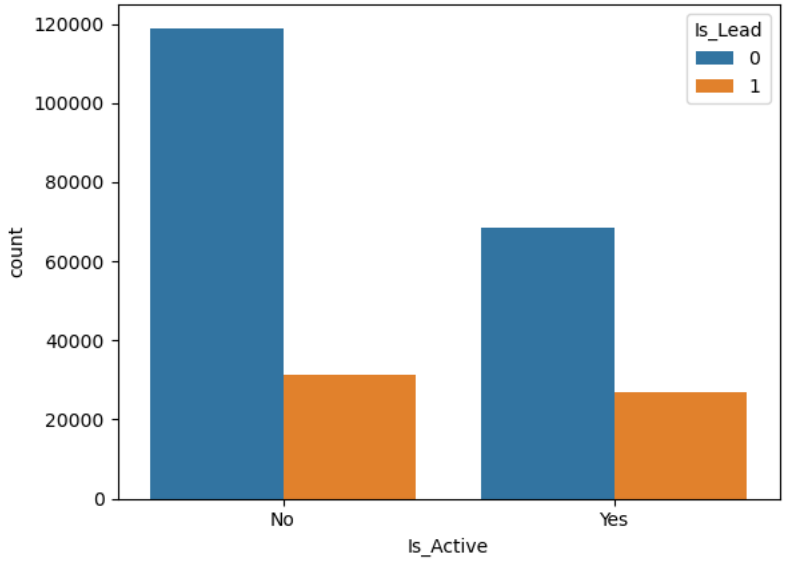
In [118]:
```
## Target Variable Distribution
plt.figure(figsize =( 7, 5))
sns.countplot(x ="Is_Lead", data=data)
plt.title("Is_Lead Distribution")
plt.show()
```



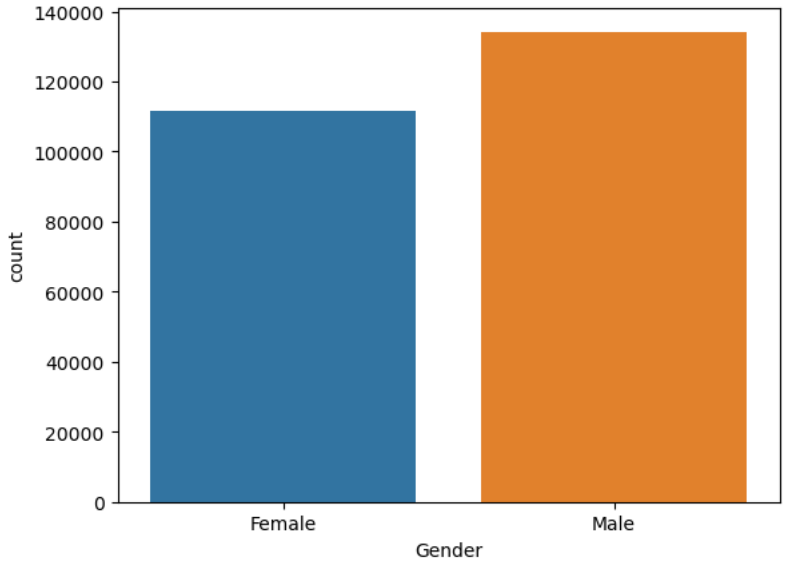
In [119]:
```
sns.countplot(x = 'Is_Active', data=data)
plt.show()
```



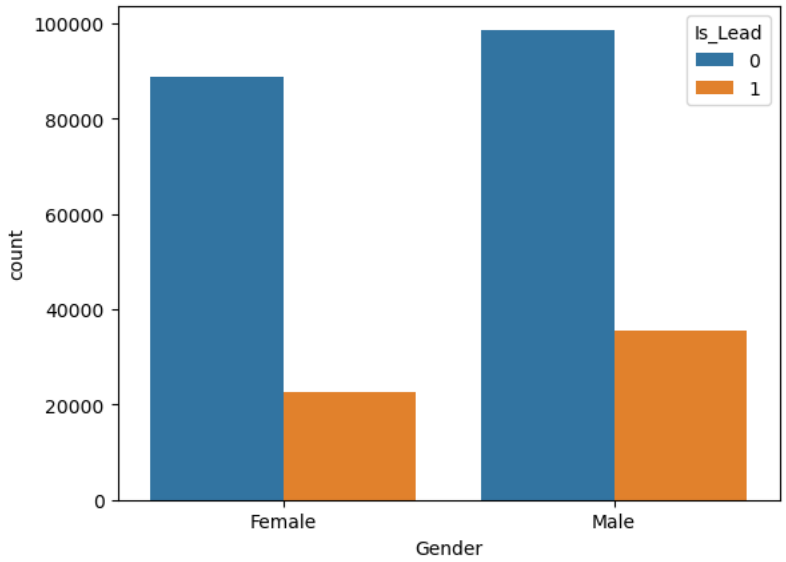
In [120]:
```
sns.countplot(x ="Is_Active", data=data, hue = 'Is_Lead')
plt.show()
```



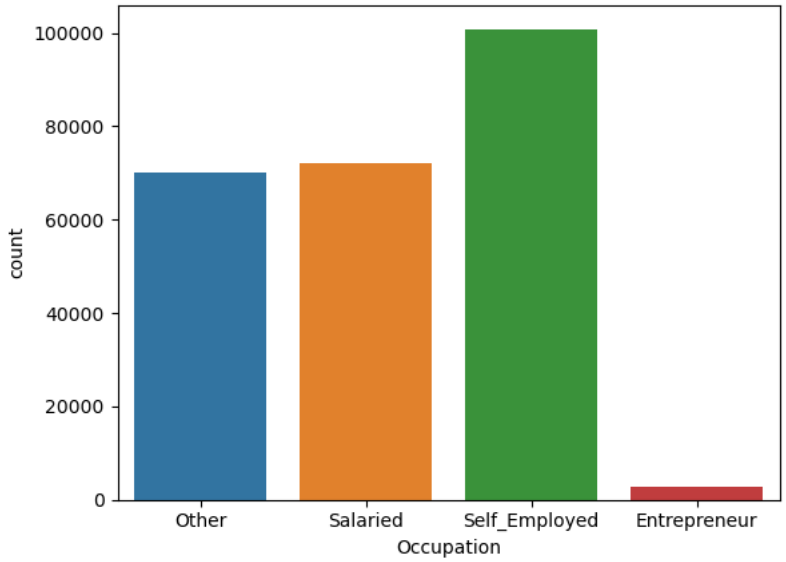
In [121]:
```
sns.countplot(x='Gender', data=data)
plt.show()
```



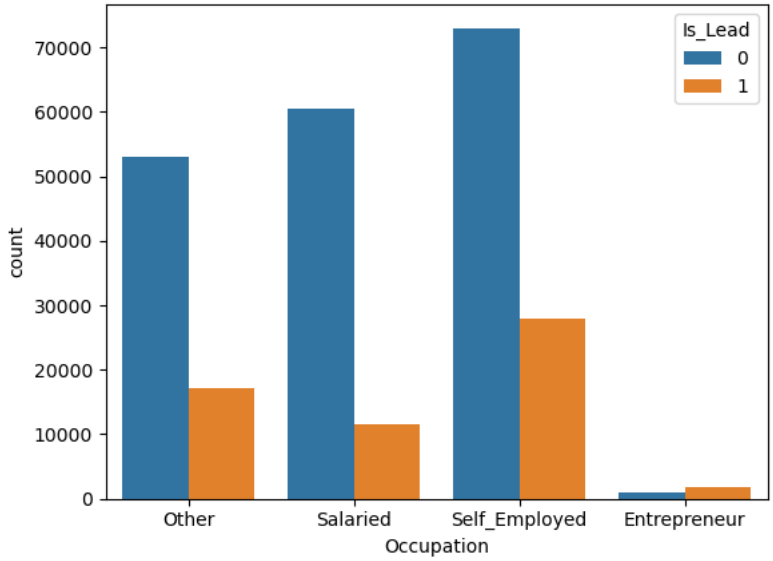
In [122]:
```
sns.countplot(x='Gender', data=data, hue='Is_Lead')
plt.show()
```



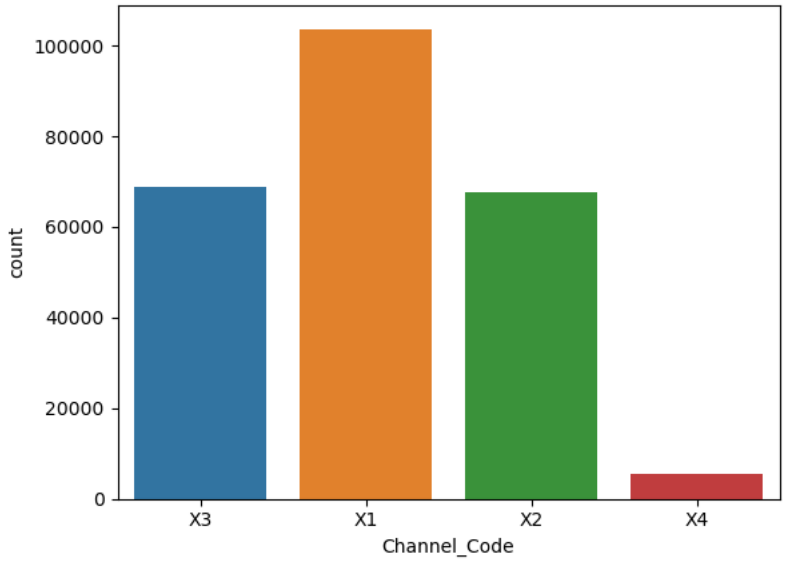
In [123]:
```
sns.countplot(x='Occupation', data=data)
plt.show()
```



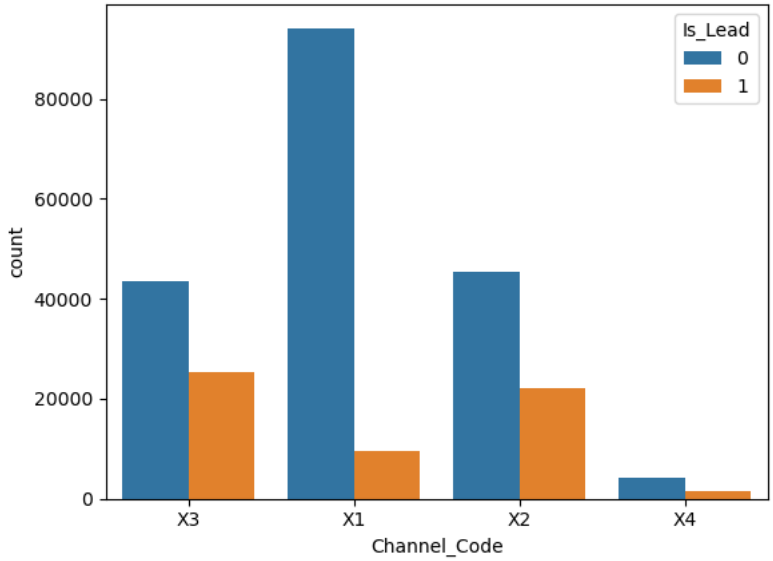
In [124]:
```
sns.countplot(x='Occupation', data=data, hue='Is_Lead')
plt.show()
```



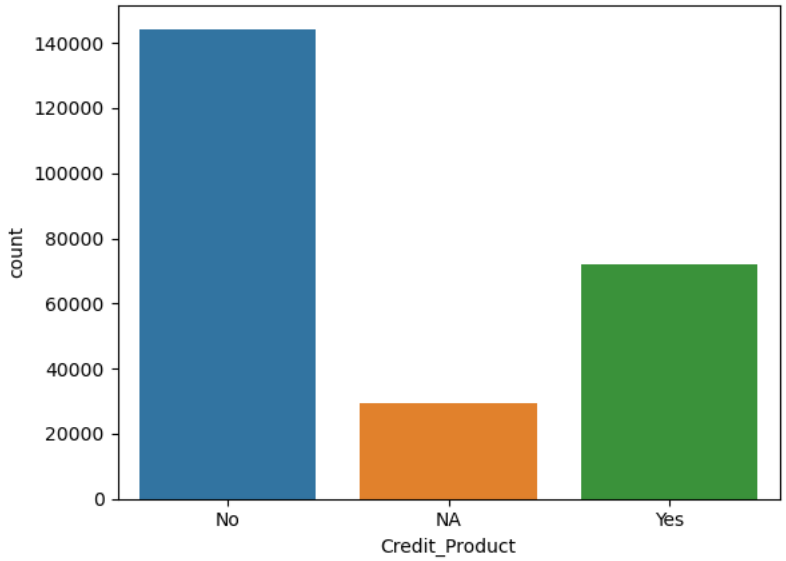
In [125]:
```
sns.countplot(x='Channel_Code', data=data)
plt.show()
```



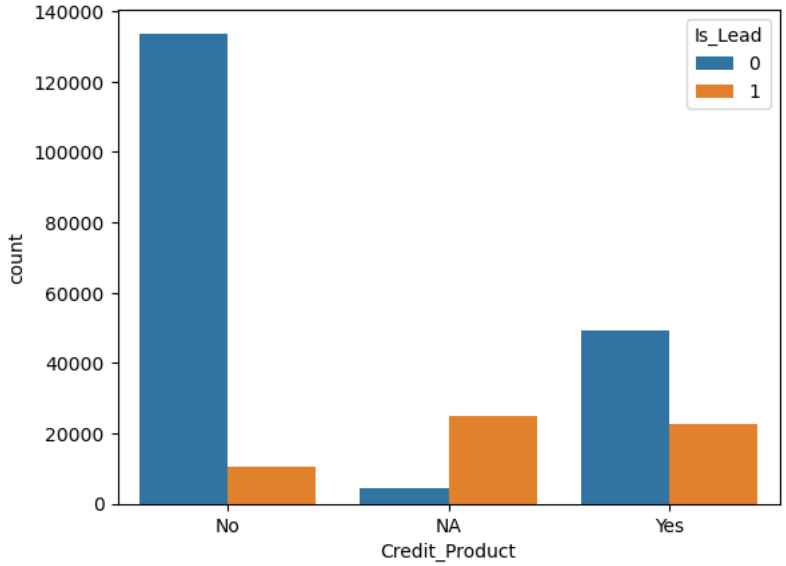
In [126]:
```
sns.countplot(x='Channel_Code', data = data, hue='Is_Lead')
plt.show()
```



In [127]:
```
sns.countplot(x='Credit_Product', data=data)
plt.show()
```

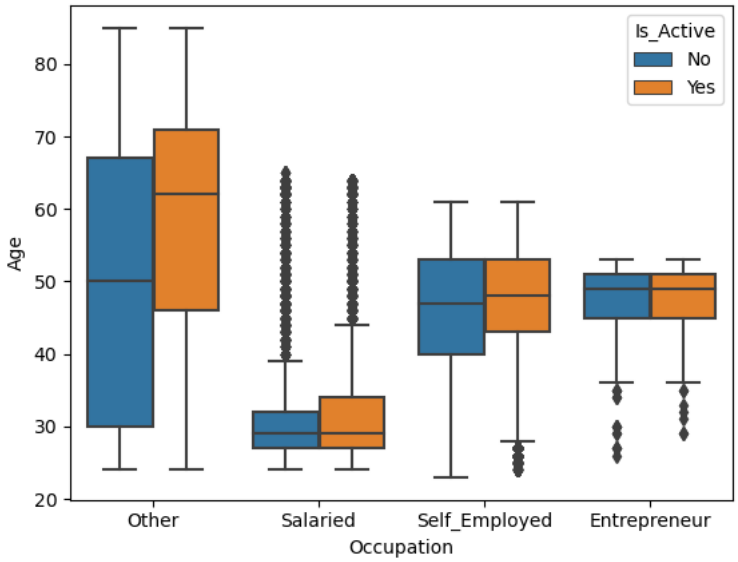


In [128]:
```
sns.countplot(x='Credit_Product', data=data, hue='Is_Lead')
plt.show()
```



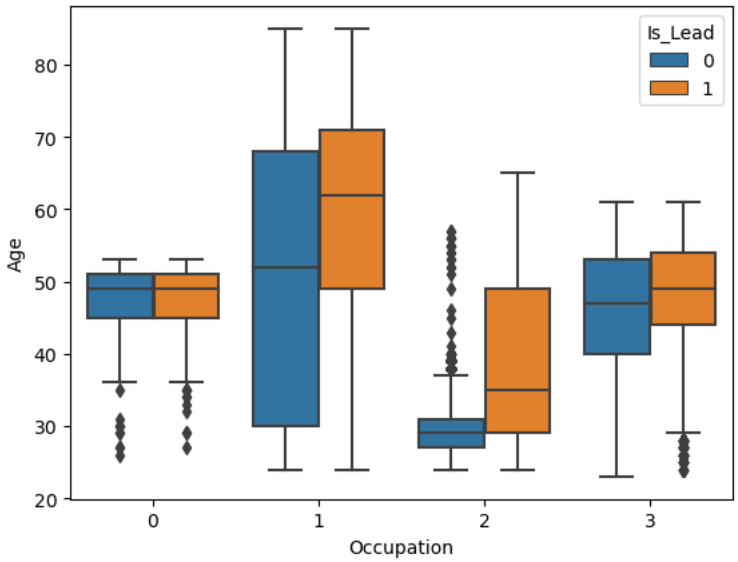
In [129]:
```
sns.boxplot(x='Occupation', y = 'Age', data = data, hue = 'Is_Active')
```

Out[129]: <Axes: xlabel='Occupation', ylabel='Age'>



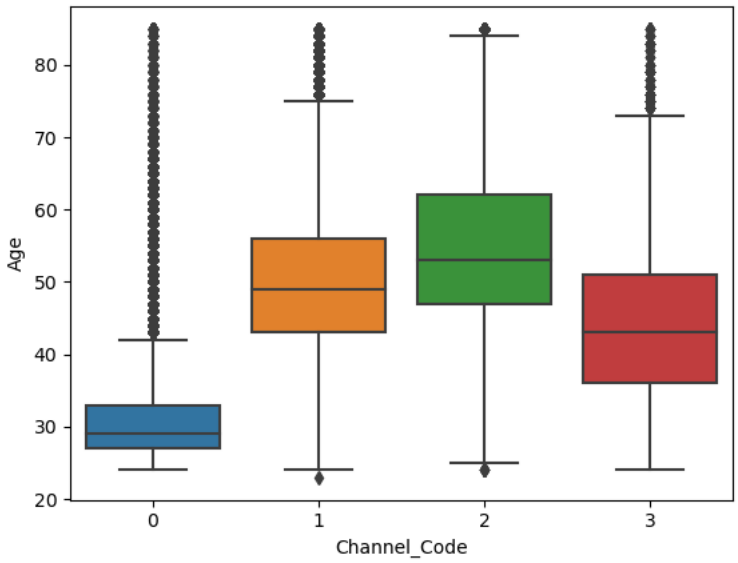
In [202]:
```
sns.boxplot(x='Occupation', y = 'Age', data = data, hue = 'Is_Lead')
```

Out[202]: <Axes: xlabel='Occupation', ylabel='Age'>



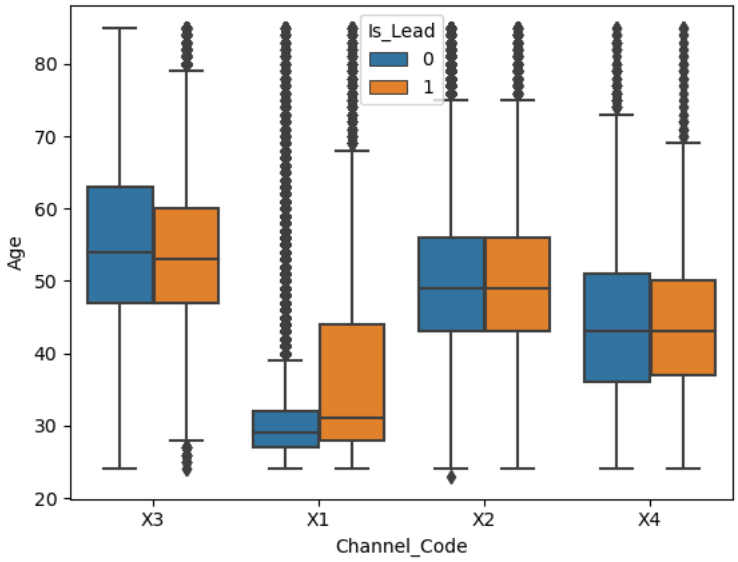
In [203]:
```
sns.boxplot(x ='Channel_Code', y = 'Age', data = data)
```

Out[203]: <Axes: xlabel='Channel_Code', ylabel='Age'>



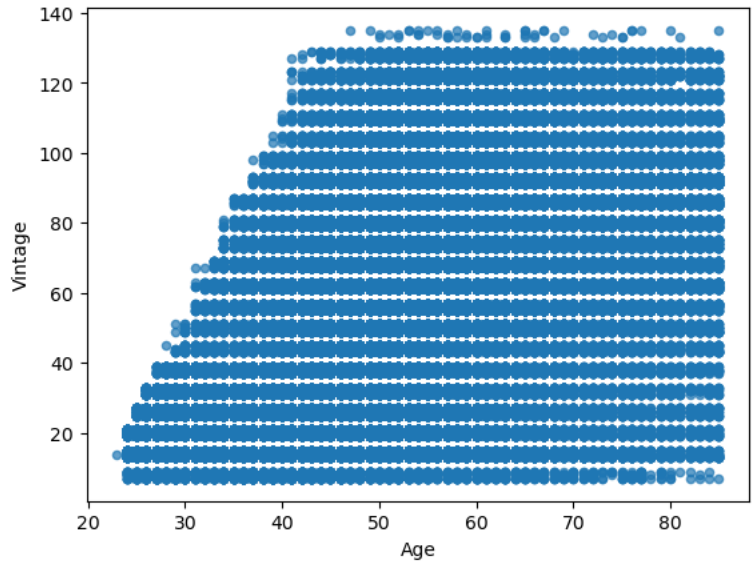
In [130]:
```
sns.boxplot(x='Channel_Code', y = 'Age', data = data, hue = 'Is_Lead')
```

Out[130]: <Axes: xlabel='Channel_Code', ylabel='Age'>



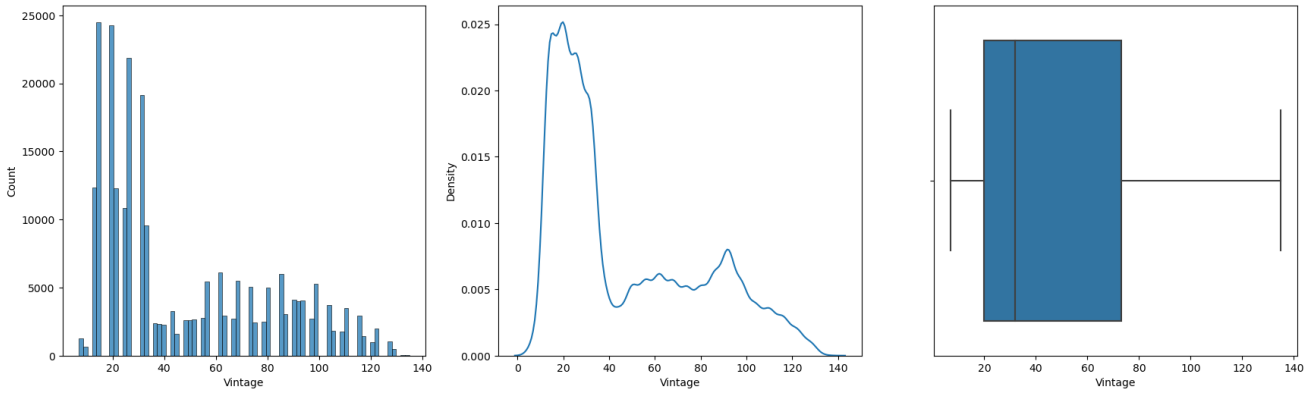
In [131]:
```
data.plot.scatter(x='Age', y = 'Vintage', alpha = 0.7)
```

Out[131]: <Axes: xlabel='Age', ylabel='Vintage'>

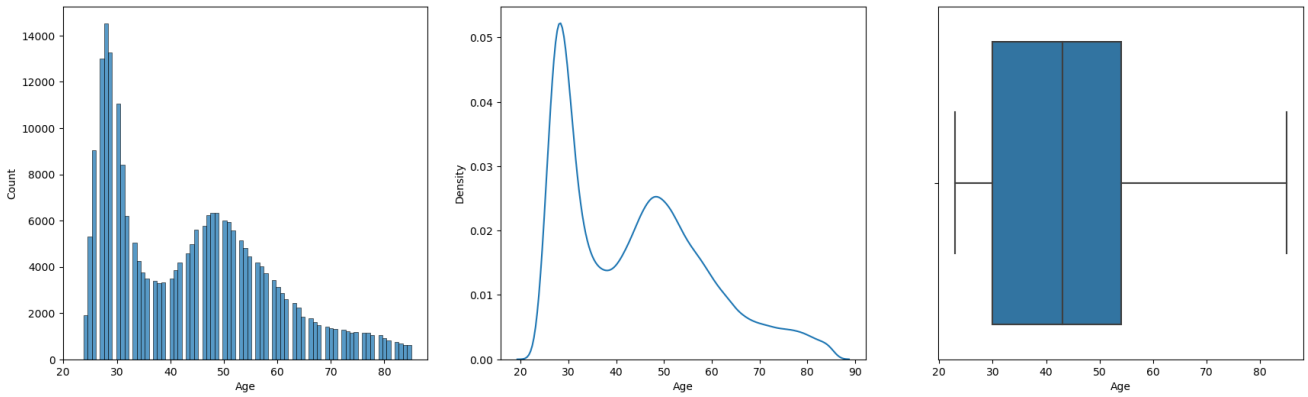


In [133]:
```
def hist_kde_box_plot(data, var: str):
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(21,6))
    sns.histplot(data, x=var, ax = axes[0])
    sns.kdeplot(data, x=var, ax = axes[1])
    sns.boxplot(data, x=var, ax= axes[2])
    plt.show()
```

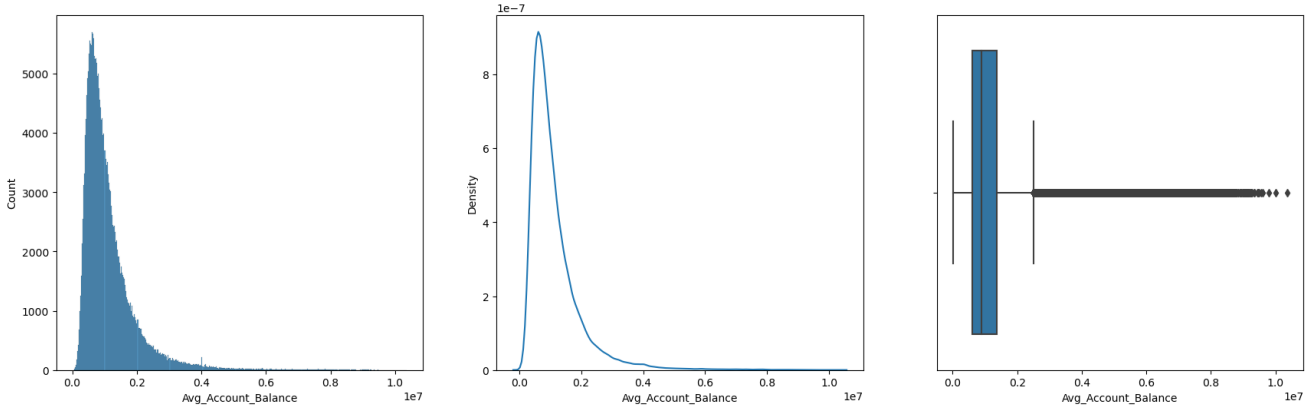
In [134]:
```
hist_kde_box_plot(data, 'Vintage')
```



In [135]:
```
hist_kde_box_plot(data, 'Age')
```



In [136]:
```
hist_kde_box_plot(data, 'Avg_Account_Balance')
```



In [137]:
```
data.Is_Active.unique()
```

Out[137]:
```
array(['No', 'Yes'], dtype=object)
```

In [138]:
```
data['Is_Active'].replace(['Yes', 'No'], [1,2], inplace = True)
```

In [139]:
```
data['Is_Active'] = data['Is_Active'].astype(float)
```

In [140]:
```
data.head()
```

Out[140]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	Female	73	RG268	Other	X3	43	No	1045696	2.0	0
1	IDD62UNG	Female	30	RG277	Salaried	X1	32	No	581988	2.0	0
2	HD3DSEMC	Female	56	RG268	Self_Employed	X3	26	No	1484315	1.0	0
3	BF3NC7KV	Male	34	RG270	Salaried	X1	19	No	470454	2.0	0
4	TEASRWXV	Female	30	RG282	Salaried	X1	33	No	886787	2.0	0

In [141]:
```
## Creating list of Categorical columns
cat_col=['Gender', 'Region_Code','Occupation', 'Channel_Code', 'Credit_Product']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in cat_col:
    data[col]=le.fit_transform(data[col])
data_1 = data
```

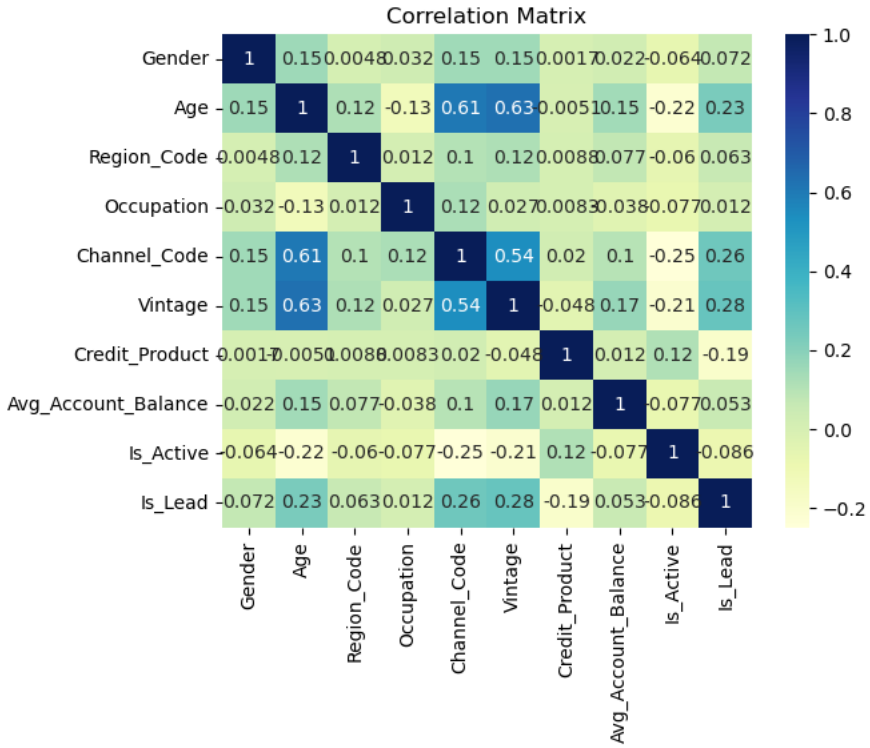
In [142]:
```
data_1.head()
```

Out[142]:

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	0	73	18	1	2	43	1	1045696	2.0	0
1	IDD62UNG	0	30	27	2	0	32	1	581988	2.0	0
2	HD3DSEMC	0	56	18	3	2	26	1	1484315	1.0	0
3	BF3NC7KV	1	34	20	2	0	19	1	470454	2.0	0
4	TEASRWXV	0	30	32	2	0	33	1	886787	2.0	0

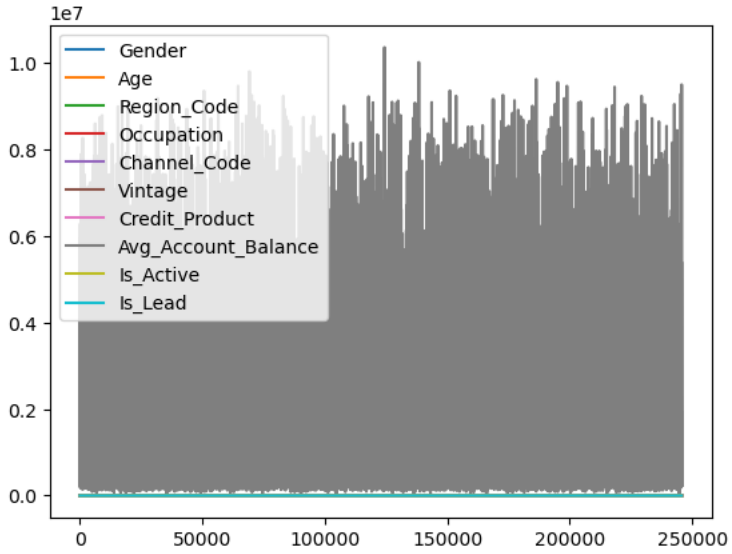
In [143]:
```
## Heatmaps help to visualize matrix-like data
## Calculating Correlations
corr = data_1.corr()
sns.heatmap(corr, cmap='YlGnBu', annot = True)
plt.title('Correlation Matrix')
plt.show()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_1720\2192494254.py:3: FutureWarning: The default value of numeric_only in DataFrame.
corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  corr = data_1.corr()
```



In [144]:
```
data.plot()
```

Out[144]: <Axes: >



In [145]:
```
data_1.drop(data_1.columns[[0,3]], axis = 1, inplace = True)
```

In [146]:
```
data_1.head()
```

Out[146]:

	Gender	Age	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	0	73	1	2	43	1	1045696	2.0	0
1	0	30	2	0	32	1	581988	2.0	0
2	0	56	3	2	26	1	1484315	1.0	0
3	1	34	2	0	19	1	470454	2.0	0
4	0	30	2	0	33	1	886787	2.0	0

In [147]:
```
X = data_1.iloc[:,[0,1,2,3,4,5,6,7]].values
y = data_1.iloc[:,-1].values
```

In [148]:
```
X,y
```

Out[148]:
```
(array([[0.000000e+00, 7.300000e+01, 1.000000e+00, ..., 1.000000e+00,
         1.045696e+06, 2.000000e+00],
        [0.000000e+00, 3.000000e+01, 2.000000e+00, ..., 1.000000e+00,
         5.819880e+05, 2.000000e+00],
        [0.000000e+00, 5.600000e+01, 3.000000e+00, ..., 1.000000e+00,
         1.484315e+06, 1.000000e+00],
        ...,
        [0.000000e+00, 2.600000e+01, 2.000000e+00, ..., 1.000000e+00,
         6.706590e+05, 2.000000e+00],
        [0.000000e+00, 2.800000e+01, 2.000000e+00, ..., 1.000000e+00,
         4.075040e+05, 2.000000e+00],
        [1.000000e+00, 2.900000e+01, 2.000000e+00, ..., 1.000000e+00,
         1.129276e+06, 2.000000e+00]]),
 array([0, 0, 0, ..., 0, 0, 0], dtype=int64))
```

In [149]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.30, random_state=0)
```

In [150]:
```
X.shape
```

Out[150]:
```
(245725, 8)
```

In [151]:
```
X_train.shape, X_test.shape
```

Out[151]:
```
((172007, 8), (73718, 8))
```

In [152]:
```
## Featuring Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [153]:
```
## Fitting Logistic Regression to the data
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

Out[153]:
```
LogisticRegression
LogisticRegression()
```

In [154]:
```
## Predicting the test results
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score, f1_score
```

In [155]:
```
y_pred = clf.predict(X_test)
print ("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.7847744105917144
```

In [156]:
```
confusion_matrix(y_test, y_pred)
```

Out[156]:
```
array([[53732,  2516],
       [13350,  4120]], dtype=int64)
```

In [157]:
```
print (classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.96      0.87     56248
           1       0.62      0.24      0.34     17470

    accuracy                           0.78     73718
   macro avg       0.71      0.60      0.61     73718
weighted avg       0.76      0.78      0.75     73718
```

In [158]:
```
print("Precision: ", precision_score(y_test, y_pred))
```

```
Precision:  0.6208559373116335
```

In [159]:
```
print("Recall: ", recall_score(y_test, y_pred))
```

```
Recall:  0.2358328563251288
```

In [160]:
```
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
F1 Score:  0.34182361237866093
```

In [161]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[161]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [162]:
```
y_pred = rf.predict(X_test)
y_pred
```

Out[162]:
```
array([0, 0, 1, ..., 1, 0, 0], dtype=int64)
```

In [163]:
```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.8423044575273338
```

In [164]:
```
confusion_matrix(y_test, y_pred)
```

Out[164]:
```
array([[52016,  4232],
       [ 7393, 10077]], dtype=int64)
```

In [165]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90     56248
           1       0.70      0.58      0.63     17470

    accuracy                           0.84     73718
   macro avg       0.79      0.75      0.77     73718
weighted avg       0.83      0.84      0.84     73718
```

In [166]:
```
print("Precision: ", precision_score(y_test, y_pred))
```

```
Precision:  0.7042420854007967
```

In [167]:
```
print("Recall: ", recall_score(y_test, y_pred))
```

```
Recall:  0.5768174012593017
```

In [168]:
```
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
F1 Score:  0.634192391201737
```

In [169]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[169]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [170]:
```
y_pred = knn.predict(X_test)
y_pred
```

Out[170]:
```
array([0, 0, 1, ..., 1, 0, 0], dtype=int64)
```

In [171]:
```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.8330122900784069
```

In [172]:
```
confusion_matrix(y_test, y_pred)
```

Out[172]:
```
array([[51435,  4813],
       [ 7497,  9973]], dtype=int64)
```

In [173]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89     56248
           1       0.67      0.57      0.62     17470

    accuracy                           0.83     73718
   macro avg       0.77      0.74      0.76     73718
weighted avg       0.83      0.83      0.83     73718
```

In [174]:
```
print("Precision: ", precision_score(y_test, y_pred))
```

```
Precision:  0.6744893818476938
```

In [175]:
```
print("Recall: ", recall_score(y_test, y_pred))
```

```
Recall:  0.5708643388666285
```

In [176]:
```
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
F1 Score:  0.6183655753968255
```

In [177]:
```
for i in range(1,8):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print("k:",i, "Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
k: 1 Accuracy Score: 0.7938359695054125
k: 2 Accuracy Score: 0.8282237716704197
k: 3 Accuracy Score: 0.8330122900784069
k: 4 Accuracy Score: 0.8432133264602947
k: 5 Accuracy Score: 0.8444477603841667
k: 6 Accuracy Score: 0.8484223663148756
k: 7 Accuracy Score: 0.8487886269296508
```

In [178]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[178]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [179]:
```
y_pred = clf.predict(X_test)
y_pred
```

Out[179]:
```
array([0, 0, 0, ..., 1, 0, 0], dtype=int64)
```

In [180]:
```
print("Accuracy Score:",accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.7869177134485472
```

In [181]:
```
confusion_matrix(y_test, y_pred)
```

Out[181]:
```
array([[48086,  8162],
       [ 7546,  9924]], dtype=int64)
```

In [182]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86     56248
           1       0.55      0.57      0.56     17470

    accuracy                           0.79     73718
   macro avg       0.71      0.71      0.71     73718
weighted avg       0.79      0.79      0.79     73718
```

In [183]:
```
print("Precision: ", precision_score(y_test, y_pred))
```

```
Precision:  0.5487117107154705
```

In [184]:
```
print("Recall: ", recall_score(y_test, y_pred))
```

```
Recall:  0.5680595306239268
```

In [185]:
```
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
F1 Score:  0.5582180222747215
```

In [186]:
```
clf_entropy = DecisionTreeClassifier(criterion='entropy')
clf_entropy.fit(X_train, y_train)
```

Out[186]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [187]:
```
y_pred = clf_entropy.predict(X_test)
y_pred
```

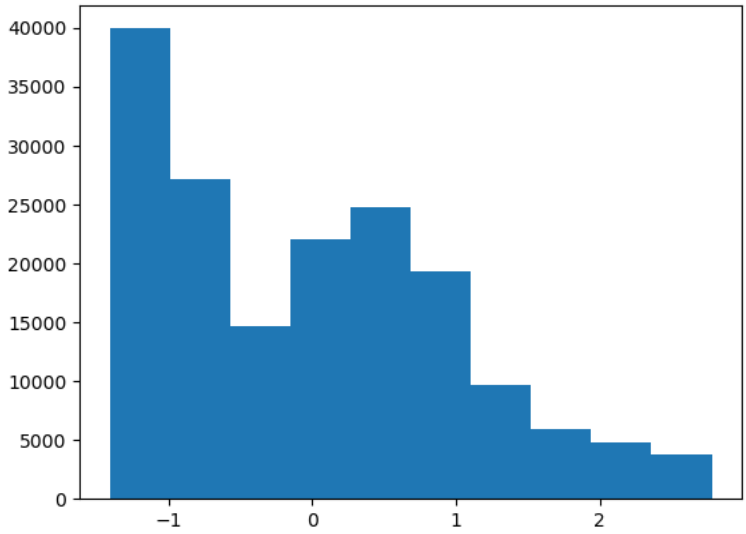
Out[187]:
```
array([0, 0, 1, ..., 1, 0, 0], dtype=int64)
```

In [188]:
```
print("Training Accuracy (Entropy):", accuracy_score(y_train, clf_entropy.predict(X_train)))
print("Test Accuracy(Entropy):", accuracy_score(y_test, y_pred))
```

```
Training Accuracy (Entropy): 0.9999825588493492
Test Accuracy(Entropy): 0.787297539271277
```

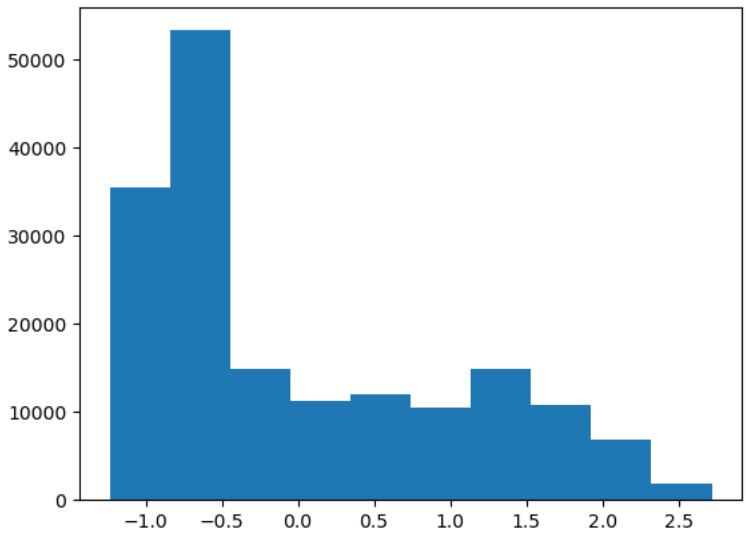
In [189]:
```
plt.hist(X_train[:,1])
```

Out[189]:
```
(array([39920., 27123., 14650., 22020., 24731., 19363.,  9662.,  5928.,
         4819.,  3791.]),
 array([-1.4074017 , -0.988901  , -0.5704003 , -0.1518996 ,  0.26660109,
         0.68510179,  1.10360249,  1.52210319,  1.94060388,  2.35910458,
         2.77760528]),
 <BarContainer object of 10 artists>)
```



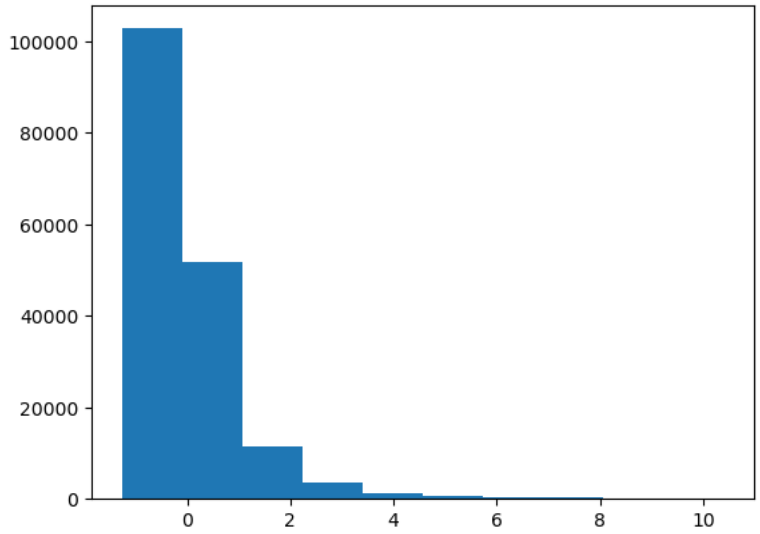
In [190]:
```
plt.hist(X_train[:,4])
```

Out[190]:
```
(array([35550., 53365., 14888., 11303., 12064., 10500., 14838., 10766.,
         6930.,  1803.]),
 array([-1.23621501, -0.84087547, -0.44553592, -0.05019638,  0.34514316,
         0.7404827 ,  1.13582224,  1.53116178,  1.92650133,  2.32184087,
         2.71718041]),
 <BarContainer object of 10 artists>)
```



In [191]:
```
plt.hist(X_train[:,6])
```

Out[191]:
```
(array([1.02836e+05, 5.17550e+04, 1.13740e+04, 3.58700e+03, 1.23600e+03,
        5.36000e+02, 3.37000e+02, 2.31000e+02, 9.00000e+01, 2.50000e+01]),
 array([-1.26807116, -0.10162269,  1.06482577,  2.23127424,  3.39772271,
         4.56417117,  5.73061964,  6.89706811,  8.06351657,  9.22996504,
        10.39641351]),
 <BarContainer object of 10 artists>)
```



In [192]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[192]:
```
GaussianNB
GaussianNB()
```

In [193]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[193]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [194]:
```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.7792126753303128
```

In [195]:
```
confusion_matrix(y_test, y_pred)
```

Out[195]:
```
array([[47054,  9194],
       [ 7082, 10388]], dtype=int64)
```

In [196]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85     56248
           1       0.53      0.59      0.56     17470

    accuracy                           0.78     73718
   macro avg       0.70      0.72      0.71     73718
weighted avg       0.79      0.78      0.78     73718
```

In [197]:
```
print("Precision: ", precision_score(y_test, y_pred))
```

```
Precision:  0.5304871821060158
```

In [198]:
```
print("Recall: ", recall_score(y_test, y_pred))
```

```
Recall:  0.5946193474527762
```

In [199]:
```
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
F1 Score:  0.5607254669113678
```

In [200]:
```
':['Logistic Regression', 'Random Forest Classifier', 'KneighborsClassifier', 'Decision Tree Classifier', 'GaussianNB Classifier'
':["0.7847", "0.8420", "0.8330", "0.7872", "0.7792"],
.6208", "0.7028", "0.6744", "0.5490", "0.5304"],
58", "0.5757", "0.5708", "0.5677", "0.5946"],
3418", "0.6329", "0.6183", "0.5582", "0.5607"]
```

In [201]:
```
df
```

Out[201]:

	Classification	Accuracy Score	Precision	Recall	F1 Score
0	Logistic Regression	0.7847	0.6208	0.2358	0.3418
1	Random Forest Classifier	0.8420	0.7028	0.5757	0.6329
2	KneighborsClassifier	0.8330	0.6744	0.5708	0.6183
3	Decision Tree Classifier	0.7872	0.5490	0.5677	0.5582
4	GaussianNB Classifier	0.7792	0.5304	0.5946	0.5607

In [ ]:
```
```