In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
train = pd.read_csv(r'train.csv')
test = pd.read_csv(r'test.csv')
```

In [3]:

```
train.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [4]:

```
test.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	

In [5]:

```
train.describe()
```

Out[5]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000

max	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

In [6]:

```
test.describe()
```

Out[6]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	367.000000	367.000000	362.000000	361.000000	338.000000
mean	4805.599455	1569.577657	136.132597	342.537396	0.825444
std	4910.685399	2334.232099	61.366652	65.156643	0.380150
min	0.000000	0.000000	28.000000	6.000000	0.000000
25%	2864.000000	0.000000	100.250000	360.000000	1.000000
50%	3786.000000	1025.000000	125.000000	360.000000	1.000000
75%	5060.000000	2430.500000	158.000000	360.000000	1.000000
max	72529.000000	24000.000000	550.000000	480.000000	1.000000

In [7]:

```
train.isnull().sum()
```

Out[7]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [8]:

```
train.shape
```

Out[8]:

```
(614, 13)
```

In [9]:

```
test.shape
```

Out[9]:

```
(367, 12)
```

In [10]:

```
train.columns
```

Out[10]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [11]:

In [11]:

```
test.columns
```

Out[11]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

In [12]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [13]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
Loan_ID              367 non-null object
Gender               356 non-null object
Married              367 non-null object
Dependents           357 non-null object
Education            367 non-null object
Self_Employed        344 non-null object
ApplicantIncome      367 non-null int64
CoapplicantIncome    367 non-null int64
LoanAmount           362 non-null float64
Loan_Amount_Term     361 non-null float64
Credit_History       338 non-null float64
Property_Area        367 non-null object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

In [14]:

```
train.duplicated().any()
```

Out[14]:

```
False
```

In [15]:

```
train.drop(['Loan_ID'], axis=1)
```

Out[15]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
0	Male	No	0	Graduate	No	5849	0.0	NaN	

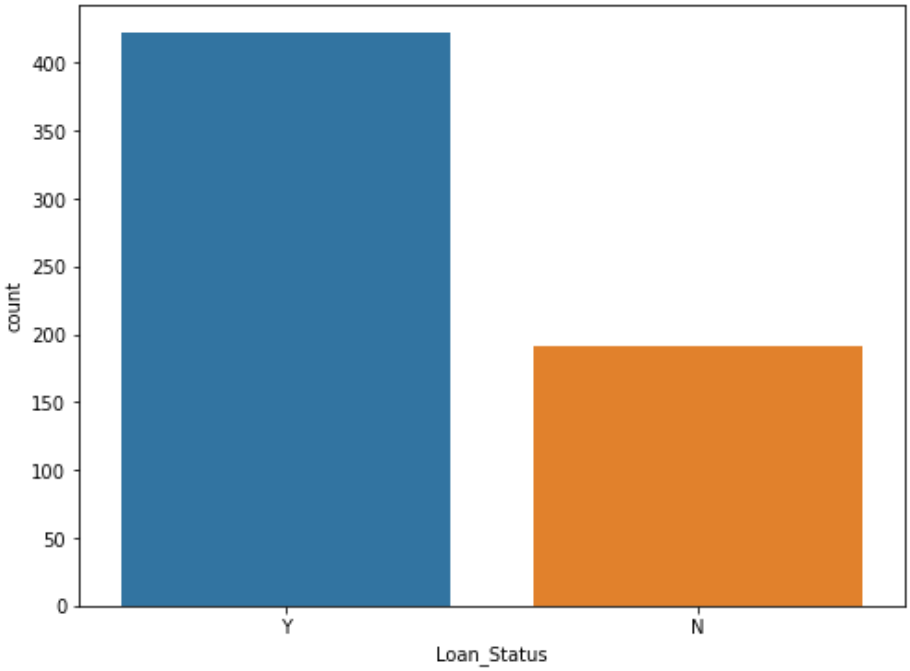
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	Male	No	0	Graduate	No	6000	0.0	141.0	
...	...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71.0	
610	Male	Yes	3+	Graduate	No	4106	0.0	40.0	
611	Male	Yes	1	Graduate	No	8072	240.0	253.0	
612	Male	Yes	2	Graduate	No	7583	0.0	187.0	
613	Female	No	0	Graduate	Yes	4583	0.0	133.0	

614 rows × 12 columns

In [16]:

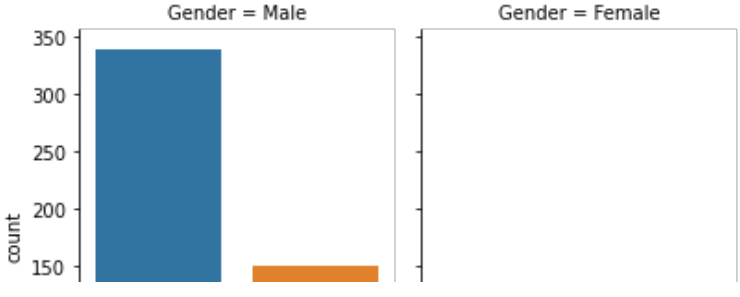
```
plt.figure(figsize=(8,6))
sns.countplot(train['Loan_Status']);

print('The percentage of Y class : %.2f' % (train['Loan_Status'].value_counts()[0] / len
(train)))
print('The percentage of N class : %.2f' % (train['Loan_Status'].value_counts()[1] / len
(train)))
```

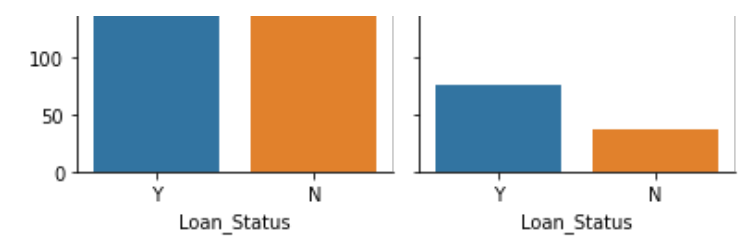
```
The percentage of Y class : 0.69
The percentage of N class : 0.31
```



In [17]:

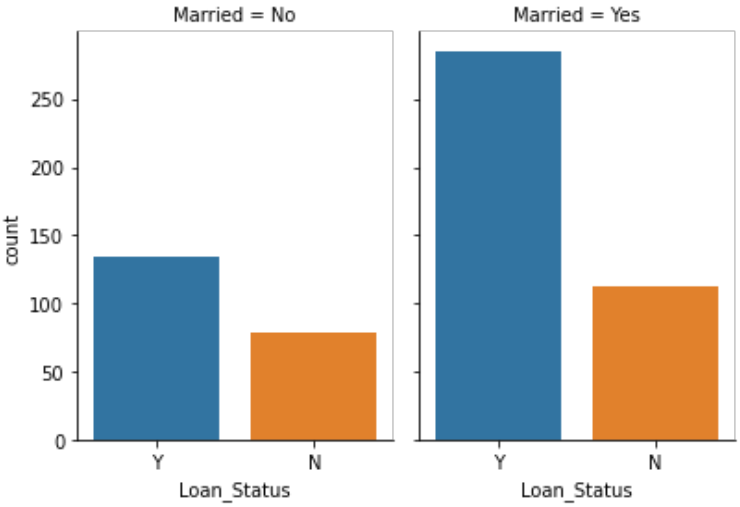
```
g = sns.catplot(x="Loan_Status", col="Gender",
                data=train, kind="count",
                height=4, aspect=.7);
```





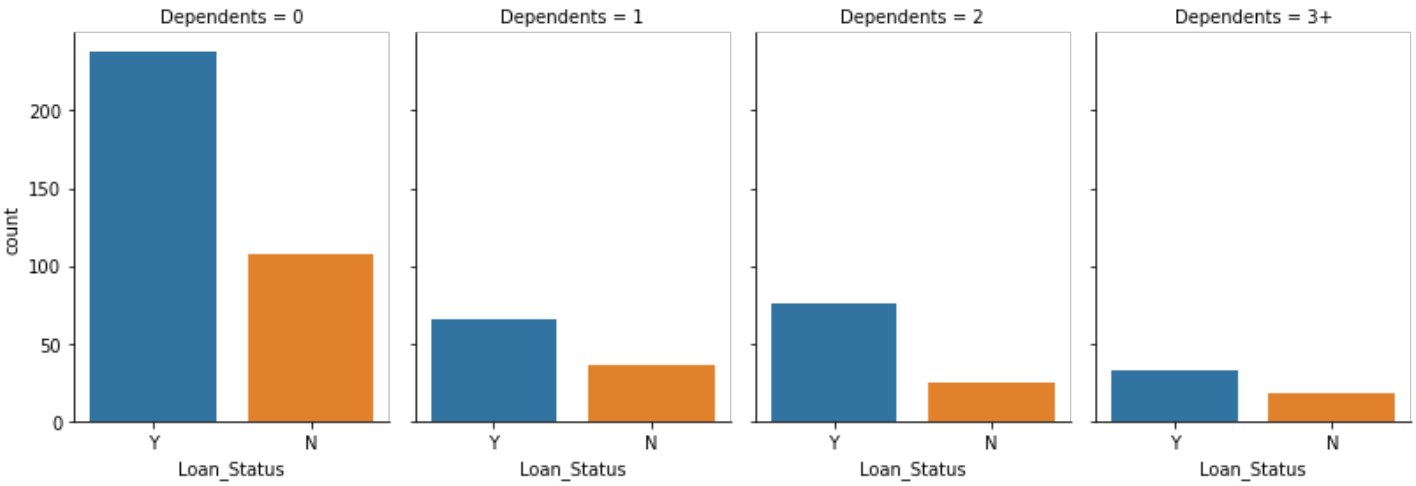
In [18]:

```
g = sns.catplot(x="Loan_Status", col='Married',
                data=train, kind="count",
                height=4, aspect=.7);
```



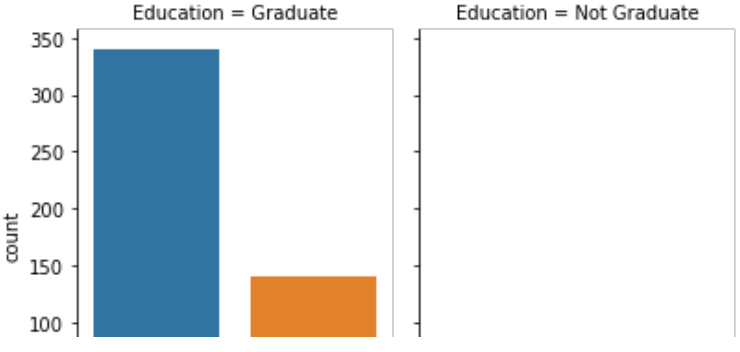
In [19]:

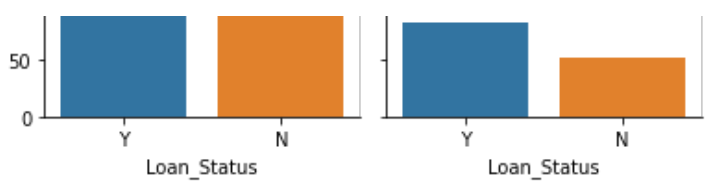
```
g = sns.catplot(x="Loan_Status", col='Dependents',
                data=train, kind="count",
                height=4, aspect=.7);
```



In [20]:

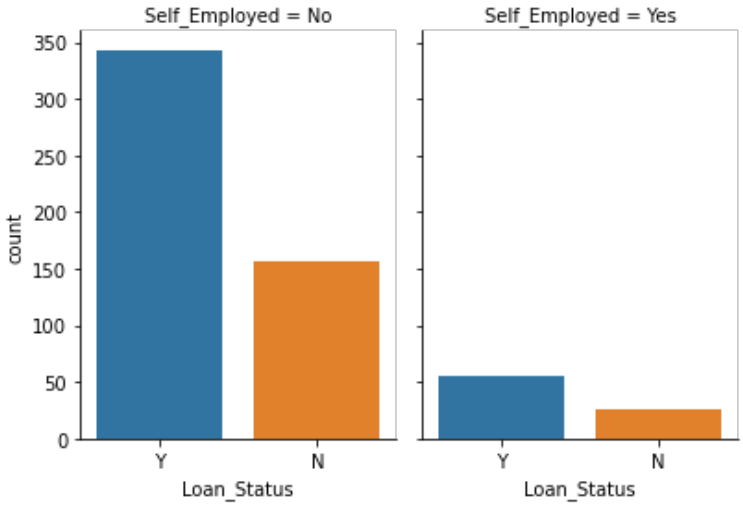
```
g = sns.catplot(x="Loan_Status", col= 'Education',
                data=train, kind="count",
                height=4, aspect=.7);
```





In [21]:

```
g = sns.catplot(x="Loan_Status", col=    'Self_Employed',
                data=train, kind="count",
                height=4, aspect=.7);
```

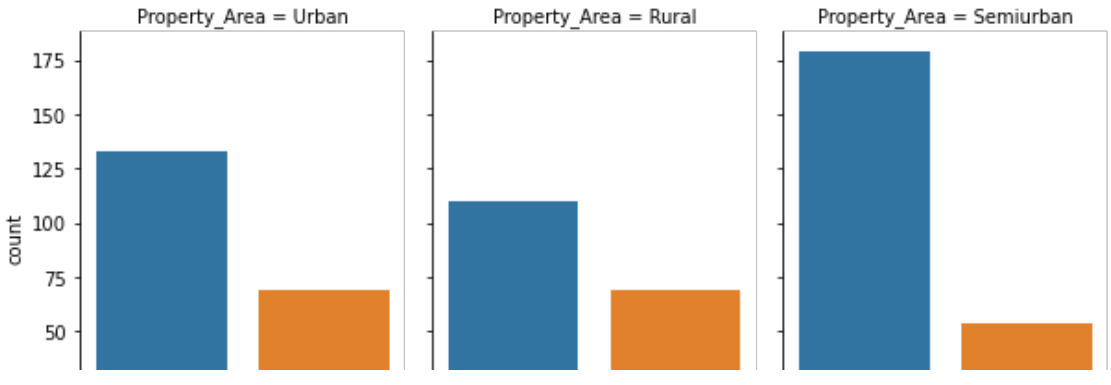


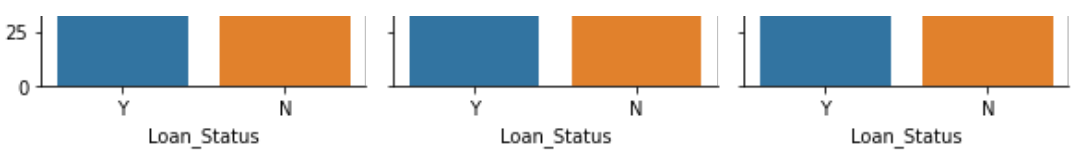
In [22]:

```
g = sns.catplot(x="Loan_Status", col = 'Credit_History',
                data=train, kind="count",
                height=4, aspect=.7);
```

In [23]:

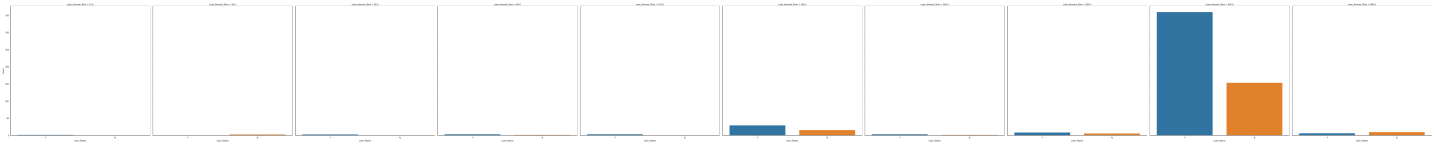
```
g = sns.catplot(x="Loan_Status", col= 'Property_Area',
                data=train, kind="count",
                height=4, aspect=.7);
```





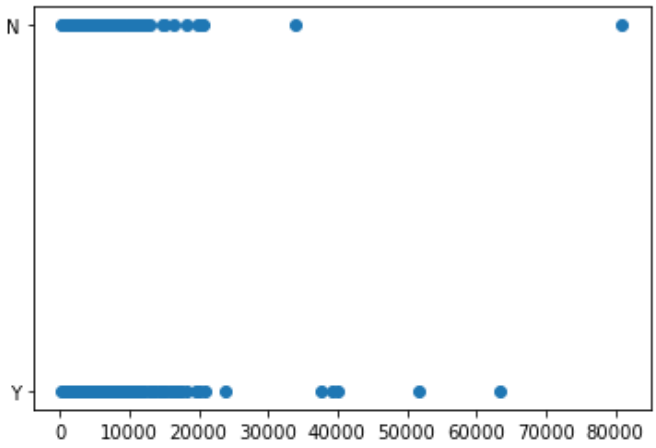
In [28]:

```
g = sns.catplot(x="Loan_Status", col= 'Loan_Amount_Term',
                data=train, kind="count",
                height=10, aspect=1);
```



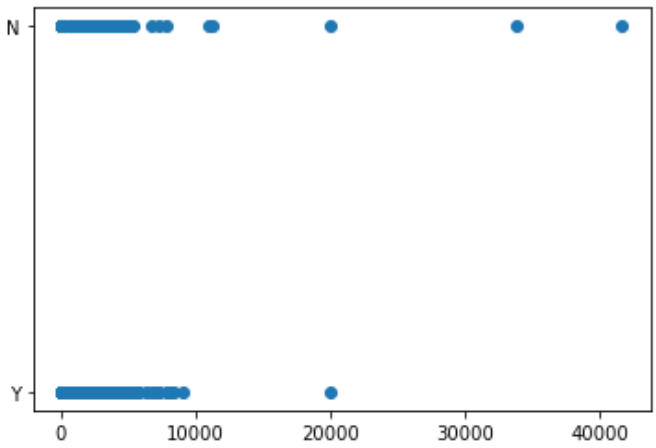
In [29]:

```
plt.scatter(train['ApplicantIncome'], train['Loan_Status']);
```



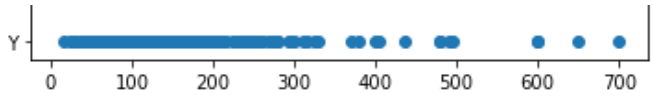
In [30]:

```
plt.scatter(train['CoapplicantIncome'], train['Loan_Status']);
```



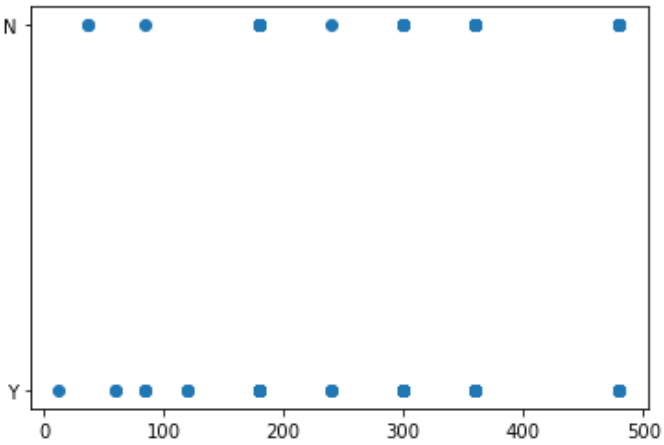
In [31]:

```
plt.scatter(train['LoanAmount'], train['Loan_Status']);
```



In [32]:

```
plt.scatter(train['Loan_Amount_Term'], train['Loan_Status']);
```



In [34]:

```
train.isnull().sum()
```

Out[34]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [35]:

```
cat_data = []
num_data = []

for i,c in enumerate(train.dtypes):
    if c == object:
        cat_data.append(train.iloc[:, i])
    else :
        num_data.append(train.iloc[:, i])
```

In [36]:

```
cat_data = pd.DataFrame(cat_data).transpose()
num_data = pd.DataFrame(num_data).transpose()
```

In [37]:

```
cat_data.head()
```

Out[37]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	Rural	N

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
2	LP001013	Male	Yes	0	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	Urban	Y
4	LP001008	Male	No	0	Graduate	No	Urban	Y

In [38]:

```
num_data.head()
```

Out[38]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	5849.0	0.0	NaN	360.0	1.0
1	4583.0	1508.0	128.0	360.0	1.0
2	3000.0	0.0	66.0	360.0	1.0
3	2583.0	2358.0	120.0	360.0	1.0
4	6000.0	0.0	141.0	360.0	1.0

In [39]:

```
#propagate non-null values forward or backward.
num_data.fillna(method = 'bfill',inplace = True)
```

In [40]:

```
num_data.isnull().sum()
```

Out[40]:

```
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
dtype: int64
```

In [41]:

```
num_data.head()
```

Out[41]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	5849.0	0.0	128.0	360.0	1.0
1	4583.0	1508.0	128.0	360.0	1.0
2	3000.0	0.0	66.0	360.0	1.0
3	2583.0	2358.0	120.0	360.0	1.0
4	6000.0	0.0	141.0	360.0	1.0

In [42]:

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
le = preprocessing.LabelEncoder()
```

In [43]:

```
cat_data = cat_data.apply(lambda col: le.fit_transform(col.astype(str)), axis=0, result_t
ype='expand')
```

In [44]:

```
for i in cat_data:
    cat_data[i] = le.fit_transform(cat_data[i])
```

In [45]:

```
cat_data.head()
```

Out[45]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	0	1	0	0	0	0	2	1
1	1	1	1	1	0	0	0	0
2	2	1	1	0	0	1	2	1
3	3	1	1	0	1	0	2	1
4	4	1	0	0	0	0	2	1

In [46]:

```
#target_values = {'Y': 0 , 'N' : 1}

target = cat_data['Loan_Status']
```

In [47]:

```
target.head()
```

Out[47]:

```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int64
```

In [48]:

```
cat_data.drop('Loan_Status', axis=1, inplace = True)
#target = target.map(target_values)
```

In [49]:

```
target.head()
```

Out[49]:

```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int64
```

In [50]:

```
cat_data.head()
```

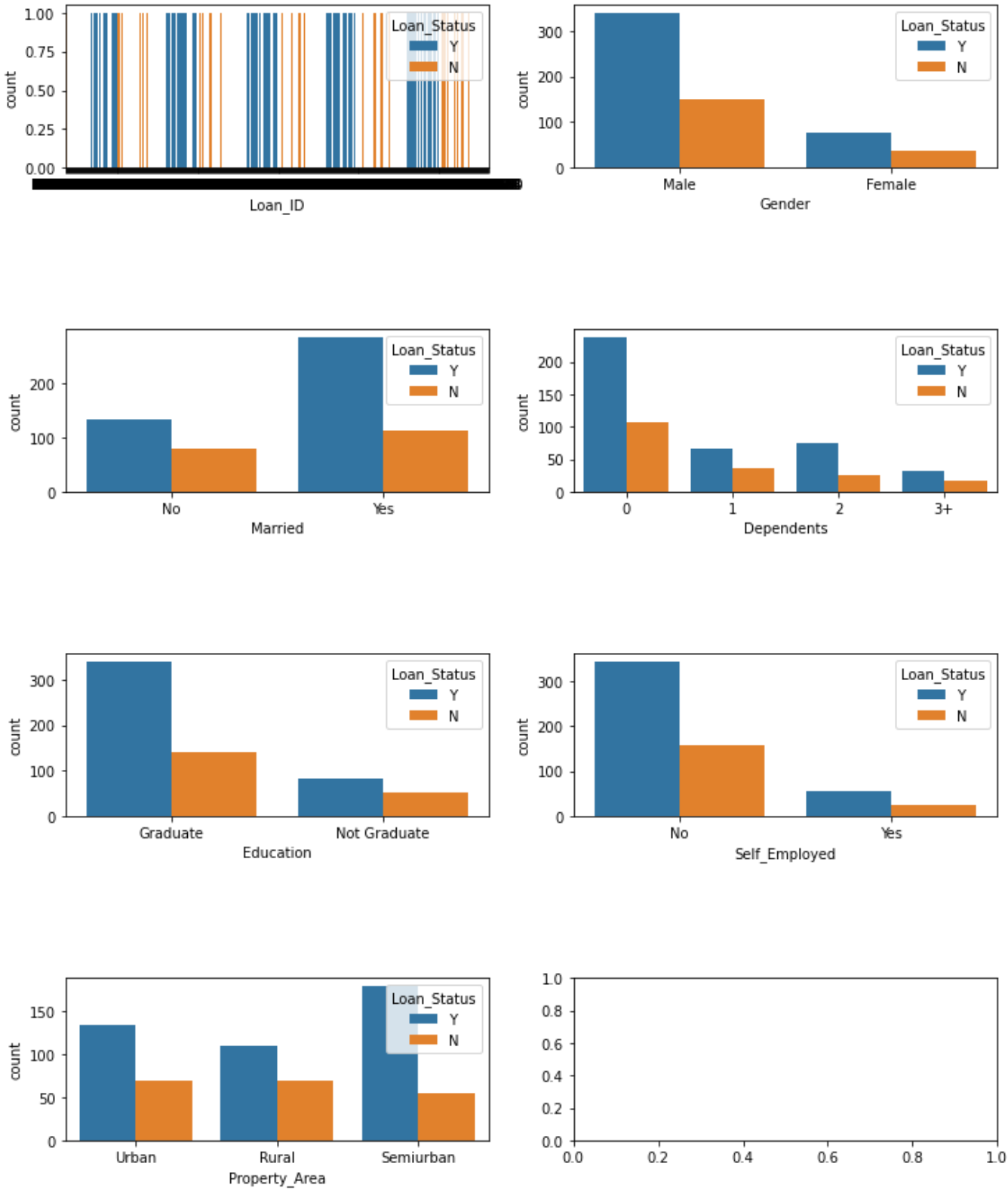
Out[50]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area
0	0	1	0	0	0	0	2
1	1	1	1	1	0	0	0
2	2	1	1	0	0	1	2
3	3	1	1	0	1	0	2

4	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area

In [51]:

```
import seaborn as sns
import matplotlib.pyplot as plt


fig,axes = plt.subplots(4,2,figsize=(12,15))
for idx,cat_col in enumerate(cat_data):
    row,col = idx//2,idx%2
    sns.countplot(x=cat_col,data=train,hue='Loan_Status',ax=axes[row,col])


plt.subplots_adjust(hspace=1)
```

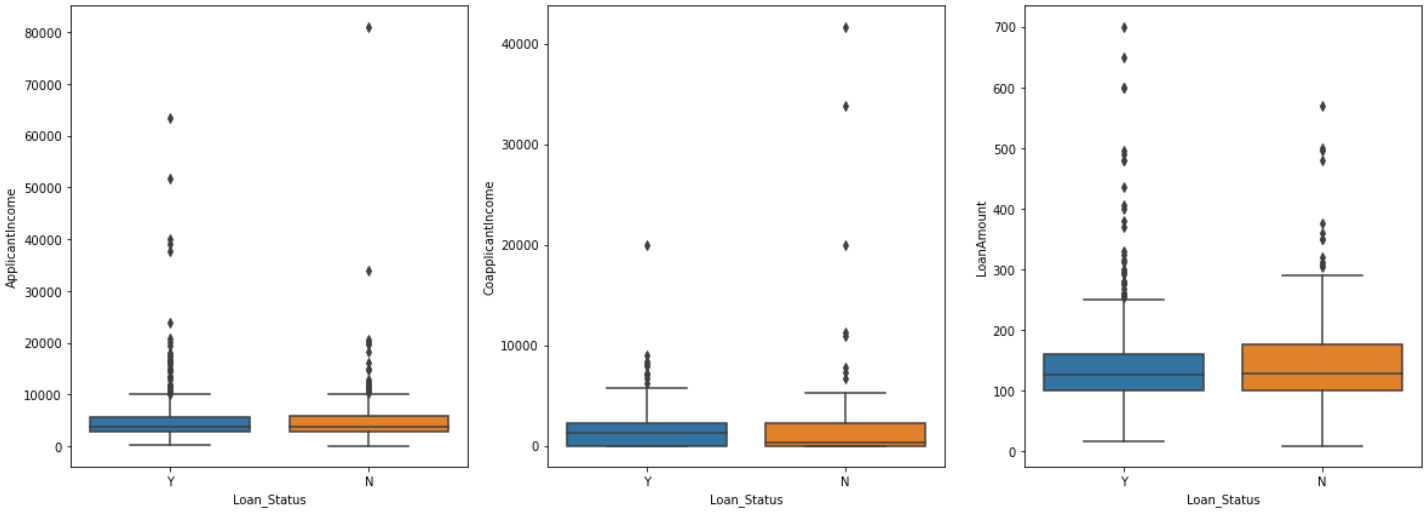


In [59]:

```
fig,axes = plt.subplots(1,3,figsize=(20,7))
```

```
for idx,cat_col in enumerate(num_data):
    sns.boxplot(y=cat_col,data=train,x='Loan_Status',ax=axes[idx])

print(train[num_data].describe())
plt.subplots_adjust(hspace=1)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-59-11179e007e4a> in <module>
      1 fig,axes = plt.subplots(1,3,figsize=(20,7))
      2 for idx,cat_col in enumerate(num_data):
----> 3     sns.boxplot(y=cat_col,data=train,x='Loan_Status',ax=axes[idx])
      4
      5 print(train[num_data].describe())

IndexError: index 3 is out of bounds for axis 0 with size 3
```



In [60]:

```
train = pd.concat([cat_data, num_data, target],axis=1)
```

In [61]:

```
train= train.drop(['Loan_ID'],axis=1)
```

In [62]:

```
train.head()
```

Out[62]:

	Gender	Married	Dependents	Education	Self_Employed	Property_Area	ApplicantIncome	CoapplicantIncome	LoanAmount
0	1	0	0	0	0	2	5849.0	0.0	128.0
1	1	1	1	0	0	0	4583.0	1508.0	128.0
2	1	1	0	0	1	2	3000.0	0.0	66.0
3	1	1	0	1	0	2	2583.0	2358.0	120.0
4	1	0	0	0	0	2	6000.0	0.0	141.0

In [63]:

```
X = pd.concat([num_data, cat_data],axis=1)
y= target
```

In [64]:

```
X.head()
```

Out[64]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_ID	Gender	Married	Dependents
0	5849.0	0.0	128.0	360.0	1.0	0	1	0	
1	4583.0	1508.0	128.0	360.0	1.0	1	1	1	
2	3000.0	0.0	66.0	360.0	1.0	2	1	1	
3	2583.0	2358.0	120.0	360.0	1.0	3	1	1	
4	6000.0	0.0	141.0	360.0	1.0	4	1	0	

In [65]:

```
y.head()
```

Out[65]:

```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int64
```

In [66]:

```
X.isnull().sum()
```

Out[66]:

```
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
Property_Area        0
dtype: int64
```

In [67]:

```
y.isnull()
```

Out[67]:

```
0      False
1      False
2      False
3      False
4      False
       ...
609    False
610    False
611    False
612    False
613    False
Name: Loan_Status, Length: 614, dtype: bool
```

In [68]:

```
from sklearn.model_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train, test in sss.split(X, y):
    X_train, X_test = X.iloc[train], X.iloc[test]
    y_train, y_test = y.iloc[train], y.iloc[test]
```

```
print('X_train shape', X_train.shape)
print('y_train shape', y_train.shape)
print('X_test shape', X_test.shape)
print('y_test shape', y_test.shape)
```

```
X_train shape (491, 12)
y_train shape (491,)
X_test shape (123, 12)
y_test shape (123,)
```

In [69]:

```
from sklearn.linear_model import LogisticRegression
```

In [70]:

```
model = LogisticRegression(solver='liblinear', random_state=0)
```

In [71]:

```
model.fit(X, y)
```

Out[71]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [72]:

```
model.predict_proba(X)
```

Out[72]:

```
array([[0.25193664, 0.74806336],
       [0.20730711, 0.79269289],
       [0.1579207 , 0.8420793 ],
       ...,
       [0.17625252, 0.82374748],
       [0.15425913, 0.84574087],
       [0.77497714, 0.22502286]])
```

In [85]:

```
model.predict(X)
```

Out[85]:

```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
```

```
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
      dtype=int64)
```

In [86]:

```
model.score(X, y)
```

Out[86]:

```
0.8045602605863192
```

In [88]:

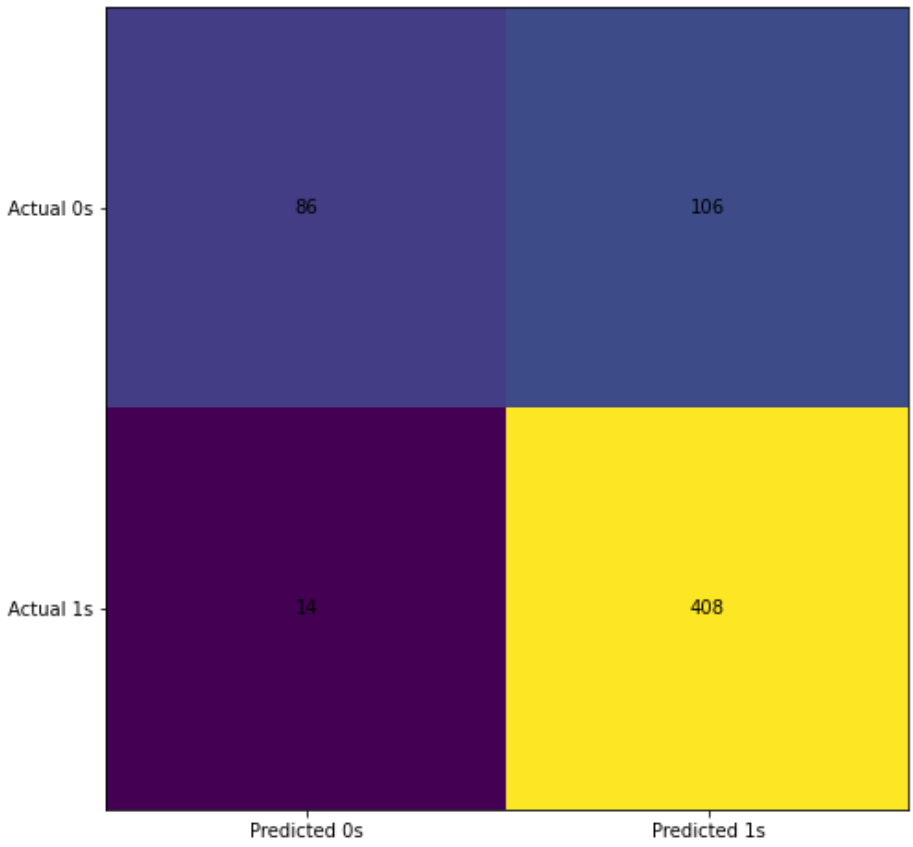
```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y, model.predict(X))
```

Out[88]:

```
array([[ 86, 106],
       [ 14, 408]], dtype=int64)
```

In [89]:

```
cm = confusion_matrix(y, model.predict(X))

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='black')
plt.show()
```



In [90]:

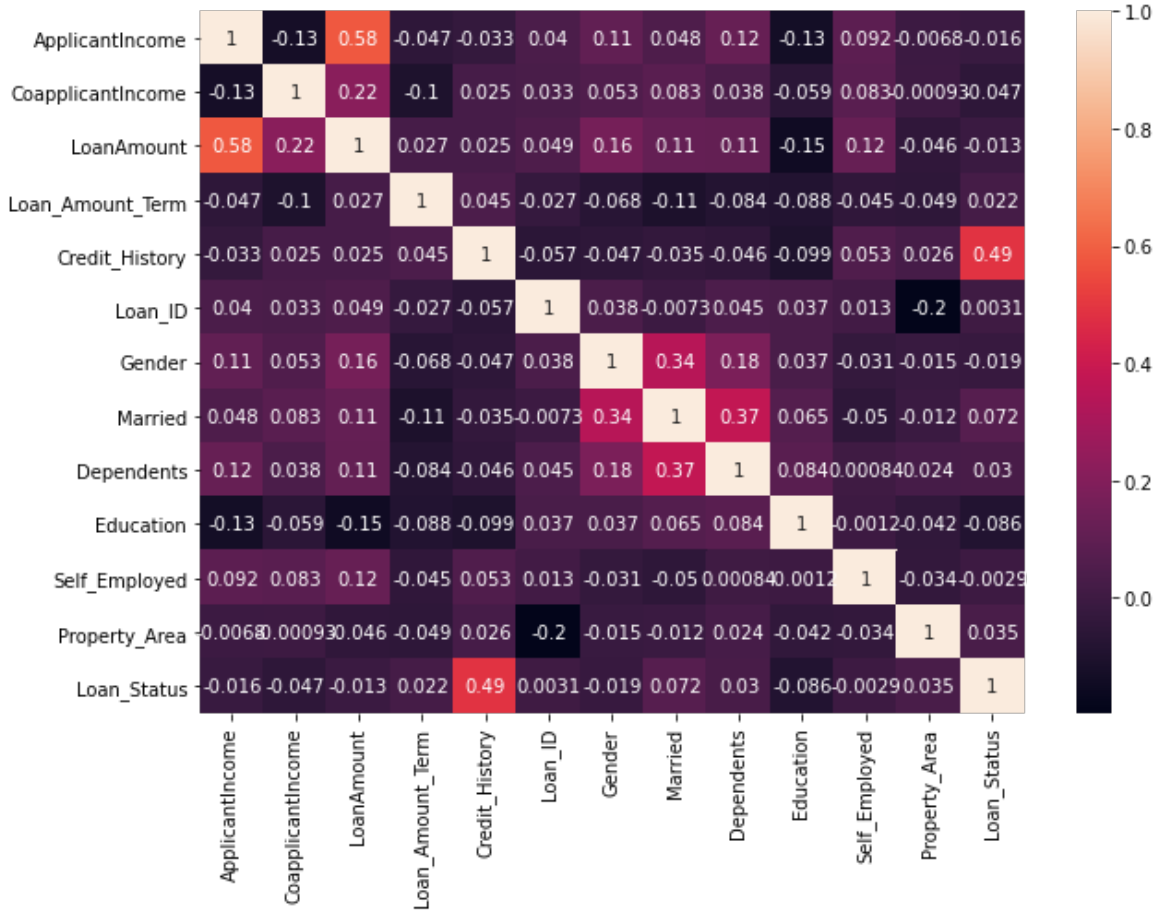
```
print(classification_report(y, model.predict(X)))
```

```
print(classification_report(y, model.predict(X)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.45      0.59       192
           1       0.79      0.97      0.87       422

    accuracy                           0.80       614
   macro avg       0.83      0.71      0.73       614
weighted avg       0.81      0.80      0.78       614
```

In [78]:

```
data_corr = pd.concat([X_train, y_train], axis=1)
corr = data_corr.corr()
plt.figure(figsize=(10,7))
sns.heatmap(corr, annot=True);
```



In [93]:

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
imp_train = imp.fit(X_train)
X_train = imp_train.transform(X_train)
X_test_imp = imp_train.transform(X_test)
```

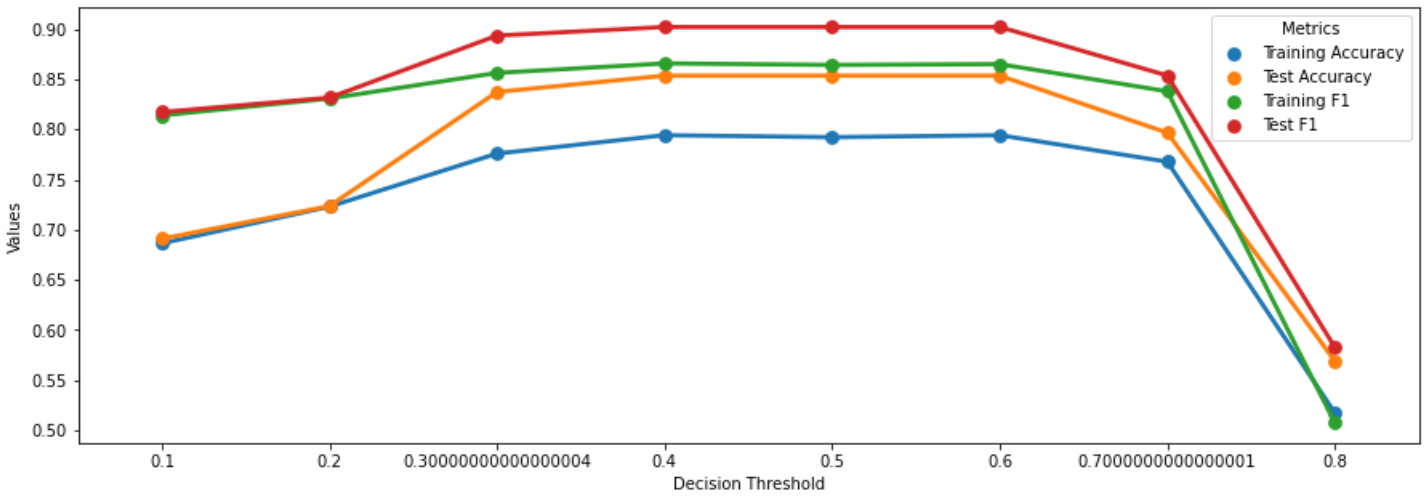
In [94]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,f1_score
from sklearn.model_selection import cross_val_predict

train_accuracies = []
train_f1_scores = []
test_accuracies = []
test_f1_scores = []
thresholds = []

#for thresh in np.linspace(0.1,0.9,8): ## Sweeping from threshold of 0.1 to 0.9
for thresh in np.arange(0.1,0.9,0.1): ## Sweeping from threshold of 0.1 to 0.9
```

```
    logreg_clf = LogisticRegression(solver='liblinear')
    logreg_clf.fit(X_train,y_train)

    y_pred_train_thresh = logreg_clf.predict_proba(X_train)[:,1]
    y_pred_train = (y_pred_train_thresh > thresh).astype(int)

    train_acc = accuracy_score(y_train,y_pred_train)
    train_f1 = f1_score(y_train,y_pred_train)

    y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
    y_pred_test = (y_pred_test_thresh > thresh).astype(int)

    test_acc = accuracy_score(y_test,y_pred_test)
    test_f1 = f1_score(y_test,y_pred_test)

    train_accuracies.append(train_acc)
    train_f1_scores.append(train_f1)
    test_accuracies.append(test_acc)
    test_f1_scores.append(test_f1)
    thresholds.append(thresh)


Threshold_logreg = {"Training Accuracy": train_accuracies, "Test Accuracy": test_accuracies, "Training F1": train_f1_scores, "Test F1":test_f1_scores, "Decision Threshold": thresholds }
Threshold_logreg_df = pd.DataFrame.from_dict(Threshold_logreg)

plot_df = Threshold_logreg_df.melt('Decision Threshold',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Decision Threshold", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[94]:

```
<AxesSubplot:xlabel='Decision Threshold', ylabel='Values'>
```



In [95]:

```
thresh = 0.4 ### Threshold chosen from above Curves
y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
y_pred = (y_pred_test_thresh > thresh).astype(int)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
Test Accuracy:  0.8536585365853658
Test F1 Score:  0.9021739130434783
Confusion Matrix on Test Data
```

Out[95]:

Predicted	0	1	All
True			

Predicted	**0**	**16**	**All**
True			
0	22	16	38
1	2	83	85
All	24	99	123

In [ ]: