Instructions

This notebook contains two questions with multiple parts. Please refer to the provided *problem statement* for these questions.

A count-up timer is located on the lower lefthand corner of this page. After 48 hours, this assessment will be automatically submitted and made read-only.

To submit your notebooks before the 48 hours have elapsed, return to https://modeling.hddatascience.us and click "Complete Course..." next to where you launched this server.

For support, please contact tech@hddatascience.us

Supplemental documents

There are three reference documents that will be used in the questions.

1. Data Dictionary.csv - a data dictionary that describes the fields of the following datasets
2. Property Level Data.csv - a dataset containing property level data
3. Census Level Data.csv - a dataset containing census level data

Review the documents. The files are found within the root notebook folder and can be loaded from code as needed.

Installing packages

You may import the packages of your chosing. Most common package are already installed on the server. If you are not able to import the package, you may install packages using `!{sys.executable} -m pip install` for python and `install.packages("forecaset")` for R within a cell as needed.

In [3]:

```
import sys
# !{sys.executable} -m pip install shap==0.38.1
```

n_jobs Hyperparameter

In case model(s) you choose require(s) setting up the `n_jobs` hyperparameter, please set `n_jobs=1` or `n_jobs=4`

Setup

If you are new to Jupyter Notebooks, please see this documentation for more information on how to run code and use the environment. Alternatively, click on the "Help" dropdown menu at the top of this page.

You may complete this notebook in either python or R. To change the kernal from python to R, go the Kernel Menu and select "Change Kernel"

Run the the cell below to import your Python packages. You may add additional packages to import as needed.

Importing Libraries

In [4]:

```
## add imports as needed
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import boxcox
from scipy import stats
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, classification_report
from sklearn.metrics import roc_curve, roc_auc_score, precision_score, recall_score, precision_recall_curve
from sklearn import metrics
from collections import Counter
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import RandomizedSearchCV
import shap
from imblearn.over_sampling import SMOTE
```

Loading Dataset

In [5]:

```
# If using python
df_main = pd.read_csv(r"Property Level Data.csv") # Load main data
df_census = pd.read_csv(r"Census Level Data.csv") # Load census data

#If using R
#df_main = read.csv('Property Level Data.csv') # Load main data
#df_census = read.csv('Census Level Data.csv') # Load census data
```

Interview Candidate Problem Statement

The Real Estate team has approached you to help predict which properties they should invest in. They have compiled market data, containing thousands of properties. It is assumed that if a property is successful, on average it will yield a $1,000,000 benefit. Conversely, it is assumed that if a property is unsuccessful, on average it will cost the business $3,000,000.

Question 1

Solution: First, let us analyse the model performace developed by the consultant The Accuracy of the model built was 0.7398 The Recall Rate of the model built was 0.9155 The Precision of the model built was 0.7190

Since the cost of investing in unsuccessful property costs the business three times the cost of the successful property, the model in use should reduce the false positives or in other words, increase the precision of the model.

The model built by the consultant has a considerably low precision value and thus we can conclude that the model's performance is poor.

Question 2

In [6]:

```
df_main['score'] = df_main['SuccesssProb'].apply(lambda x: 1 if x > 0.7398 else 0)
df_main
```

Out[6]:

	PropertyId	StateCode	BuildingCount	StoryCount	YearBuilt	UnitCount	NetRentableSF	YearLastRenovated	ParkingRatio	Gro
0	1	0	2	3	1999	90	55384.10083	-1	Over 200%	
1	2	0	2	4	1991	102	132096.75190	-1	Between 150% and 200%	

	PropertyId	StateCode	BuildingCount	StoryCount	YearBuilt	UnitCount	NetRentableSF	YearLastRenovated	ParkingRatio	Gro
2	3	0	2	1	1993	22	15771.24824	-1	Under 25%	
3	4	0	2	4	1973	124	87231.90214	-1	Between 25% and 150%	
4	5	0	2	4	2006	92	122217.85050	-1	Between 150% and 200%	
...	...	...	...	...	...	...	...	...	...	
48014	59995	11	2	4	1995	125	162024.19590	-1	Under 25%	
48015	59996	11	3	1	1969	51	90402.64113	1976	Between 150% and 200%	
48016	59998	11	2	2	1997	60	75382.01620	2009	Over 200%	
48017	59999	11	5	5	1965	320	398295.57020	-1	Between 25% and 150%	
48018	60000	11	4	23	1980	1031	875727.69910	-1	Under 25%	

48019 rows × 22 columns

Since the vendor's predictions cannot be used from now onwards, we will add a new feature called score that will be used for our classification and drop successprob feature.

Check for Null/Non-null values

In [7]:

```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48019 entries, 0 to 48018
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   PropertyId          48019 non-null  int64
 1   StateCode           48019 non-null  int64
 2   BuildingCount       48019 non-null  int64
 3   StoryCount          48019 non-null  int64
```

```
 4   YearBuilt            48019 non-null  int64
 5   UnitCount            48019 non-null  int64
 6   NetRentableSF        48019 non-null  float64
 7   YearLastRenovated    48019 non-null  int64
 8   ParkingRatio         48019 non-null  object
 9   GrossLandArea        48019 non-null  float64
 10  PropertyType         48019 non-null  object
 11  PropertySubType      48019 non-null  object
 12  OccupancyPercentage  48019 non-null  float64
 13  AnnualAverageRent    12009 non-null  float64
 14  PropertyValue        22752 non-null  float64
 15  ExpenseTax           2452 non-null   float64
 16  ExpenseRepairs       2452 non-null   float64
 17  ExpenseInsurance     2452 non-null   float64
 18  ExpensePayroll       2452 non-null   float64
 19  ExpenseGeneralFees   2452 non-null   float64
 20  SuccesssProb         48019 non-null  float64
 21  score                48019 non-null  int64
dtypes: float64(11), int64(8), object(3)
memory usage: 8.1+ MB
```

In [8]:

```
df_main.isna().sum()
```

Out[8]:

```
PropertyId                 0
StateCode                  0
BuildingCount              0
StoryCount                 0
YearBuilt                  0
UnitCount                  0
NetRentableSF              0
YearLastRenovated          0
ParkingRatio               0
GrossLandArea              0
PropertyType               0
PropertySubType            0
OccupancyPercentage        0
AnnualAverageRent      36010
PropertyValue          25267
ExpenseTax             45567
ExpenseRepairs         45567
ExpenseInsurance       45567
ExpensePayroll         45567
ExpenseGeneralFees     45567
SuccesssProb               0
```

```
score                        0
dtype: int64
```

Check percentage of null values

In [9]:

```
percent_missing = df_main.isnull().sum() * 100 / len(df_main)
print(percent_missing)
```

```
PropertyId              0.000000
StateCode               0.000000
BuildingCount           0.000000
StoryCount              0.000000
YearBuilt               0.000000
UnitCount               0.000000
NetRentableSF           0.000000
YearLastRenovated       0.000000
ParkingRatio            0.000000
GrossLandArea           0.000000
PropertyType            0.000000
PropertySubType         0.000000
OccupancyPercentage     0.000000
AnnualAverageRent      74.991149
PropertyValue          52.618755
ExpenseTax             94.893688
ExpenseRepairs         94.893688
ExpenseInsurance       94.893688
ExpensePayroll         94.893688
ExpenseGeneralFees     94.893688
SuccesssProb            0.000000
score                   0.000000
dtype: float64
```

Since we have a high majority of missing values (all above 50%), I will be dropping the respective columns

In [10]:

```
df1 = df_main.copy()
df1 = df1.drop(['AnnualAverageRent','PropertyValue','ExpenseTax','ExpenseRepairs','ExpenseInsurance','ExpensePa
df1 = df1.reset_index(drop=True)
```

In [11]:

```
df1
```

Out[11]:

	PropertyId	StateCode	BuildingCount	StoryCount	YearBuilt	UnitCount	NetRentableSF	YearLastRenovated	ParkingRatio	Gro

	PropertyId	StateCode	BuildingCount	StoryCount	YearBuilt	UnitCount	NetRentableSF	YearLastRenovated	ParkingRatio	Gro
0	1	0	2	3	1999	90	55384.10083	-1	Over 200%	
1	2	0	2	4	1991	102	132096.75190	-1	Between 150% and 200%	
2	3	0	2	1	1993	22	15771.24824	-1	Under 25%	
3	4	0	2	4	1973	124	87231.90214	-1	Between 25% and 150%	
4	5	0	2	4	2006	92	122217.85050	-1	Between 150% and 200%	
...	...	...	...	...	...	...	...	...	...	
48014	59995	11	2	4	1995	125	162024.19590	-1	Under 25%	
48015	59996	11	3	1	1969	51	90402.64113	1976	Between 150% and 200%	
48016	59998	11	2	2	1997	60	75382.01620	2009	Over 200%	
48017	59999	11	5	5	1965	320	398295.57020	-1	Between 25% and 150%	
48018	60000	11	4	23	1980	1031	875727.69910	-1	Under 25%	

48019 rows × 15 columns

In [12]:

```
df1 = df1.drop(['SuccesssProb'],axis=1)
df1 = df1.reset_index(drop=True)
```

In [13]:

```
print('Shape before and after dropping NA columns are {} and {} respectively'.format(df_main.shape, df1.shape))
```

```
Shape before and after dropping NA columns are (48019, 22) and (48019, 14) respectively
```

In [14]:

```
df1['PR'] = df1['ParkingRatio'].apply(lambda x: x.split(' ')[1].split('%')[0]).astype('int')
df1['YLT'] = np.where(df1['YearLastRenovated'] == -1, df1['YearBuilt'], df1['YearLastRenovated'])
```

Replacing -1 value with the year the property was built (assuming for properties with renovation, the year at which it was built was considered to be the year it was renovated).

In [15]:

```
df1 = df1.drop(['ParkingRatio','YearLastRenovated'],axis=1)
df1 = df1.reset_index(drop=True)
```

In [16]:

```
df1
```

Out[16]:

	PropertyId	StateCode	BuildingCount	StoryCount	YearBuilt	UnitCount	NetRentableSF	GrossLandArea	PropertyType	Proper
0	1	0	2	3	1999	90	55384.10083	16.00	Multifamily	
1	2	0	2	4	1991	102	132096.75190	13.39	Multifamily	
2	3	0	2	1	1993	22	15771.24824	2.09	Multifamily	
3	4	0	2	4	1973	124	87231.90214	0.66	Multifamily	
4	5	0	2	4	2006	92	122217.85050	4.30	Multifamily	
...	...	...	...	...	...	...	...	...	...	
48014	59995	11	2	4	1995	125	162024.19590	38.00	Multifamily	
48015	59996	11	3	1	1969	51	90402.64113	0.00	Multifamily	
48016	59998	11	2	2	1997	60	75382.01620	9.00	Multifamily	
48017	59999	11	5	5	1965	320	398295.57020	0.00	Multifamily	
48018	60000	11	4	23	1980	1031	875727.69910	0.00	Multifamily	

48019 rows × 14 columns

In [17]:

```
df1.nunique(axis=0)
```

Out[17]:

```
PropertyId           48019
StateCode               12
BuildingCount            5
StoryCount              39
YearBuilt              119
UnitCount              989
NetRentableSF        48019
GrossLandArea         1222
PropertyType             1
```

```
PropertySubType            9
OccupancyPercentage    47883
score                      2
PR                         3
YLT                      121
dtype: int64
```

Splitting the data to continuous and discrete to check distribution

In [18]:

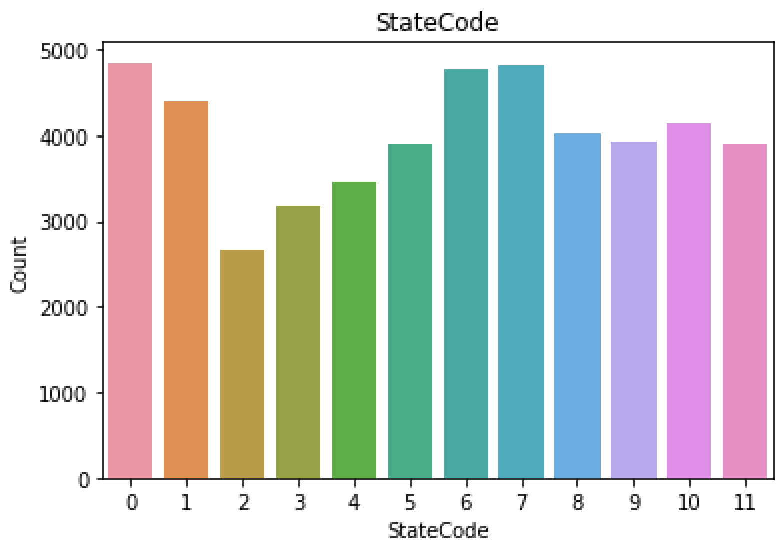
```
discrete_features=[feature for feature in df1.columns if len(df1[feature].unique())<=12 ]
print("Discrete variables count is {}".format(len(discrete_features)))
print(discrete_features)
```

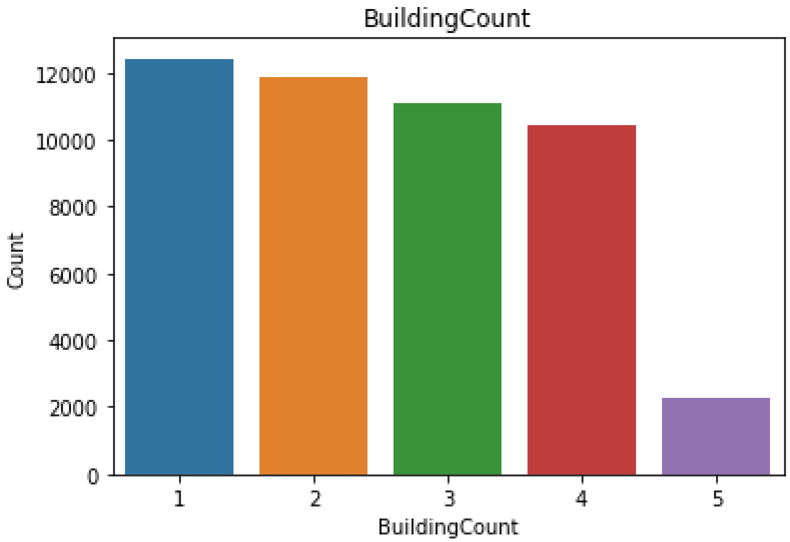
```
Discrete variables count is 6
['StateCode', 'BuildingCount', 'PropertyType', 'PropertySubType', 'score', 'PR']
```

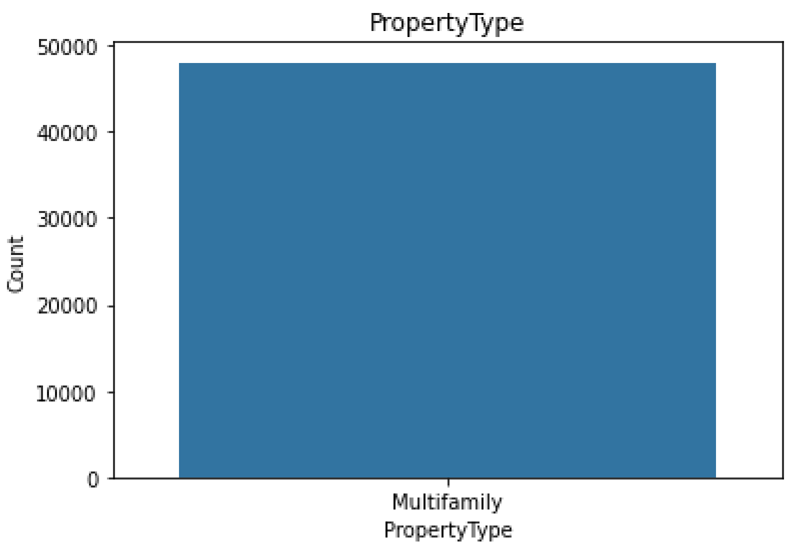
Plotting frequency distribution for discrete variables

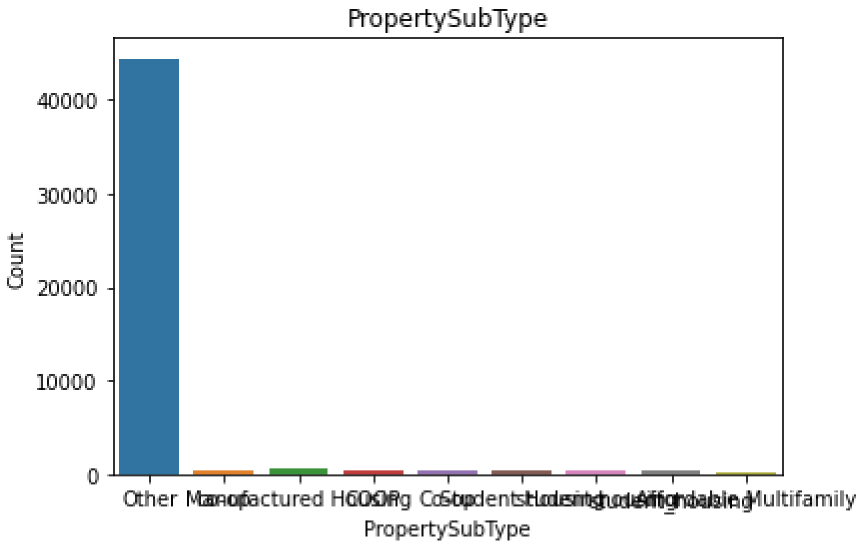
In [19]:

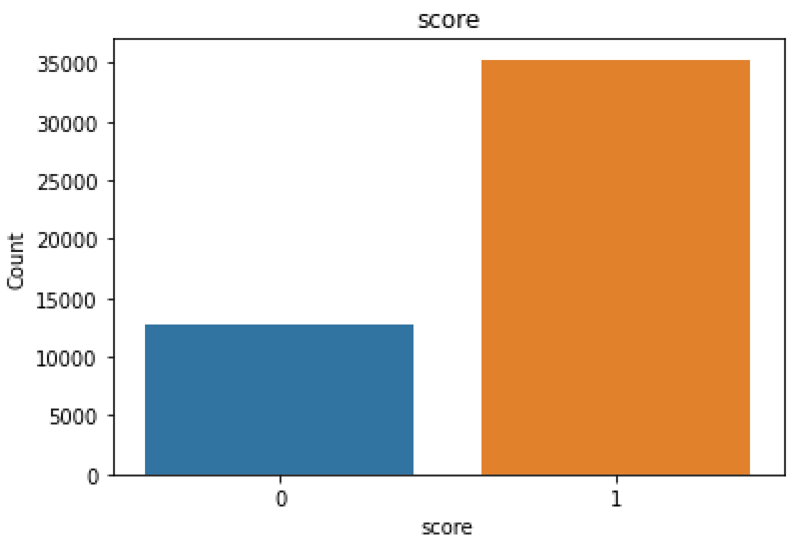
```
for feature in discrete_features:
    sns.countplot(x = feature, data = df1)
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

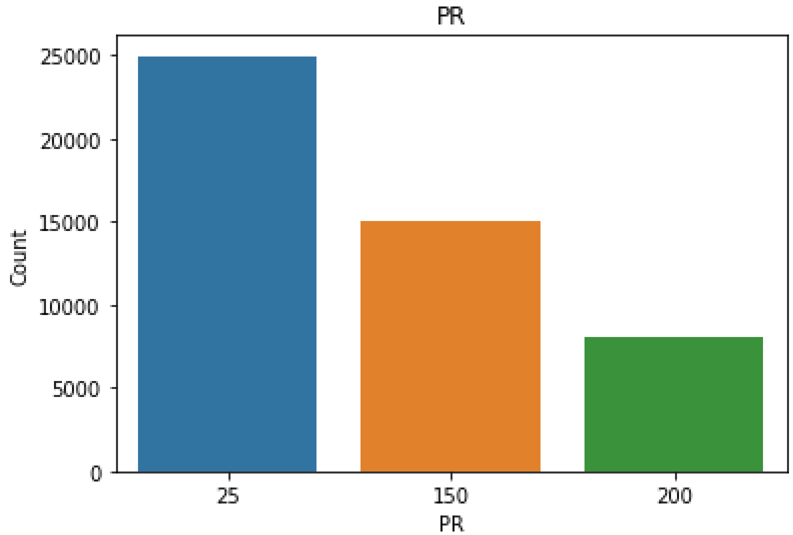



BuildingCount
Count
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
BuildingCount


PropertyType
Count
50000
40000
30000
20000
10000
0
Multifamily
PropertyType


PropertySubType
Count
40000
30000
20000
10000
0
PropertySubType


score
Count
35000
30000
25000
20000
15000
10000
5000
0
0
1
score



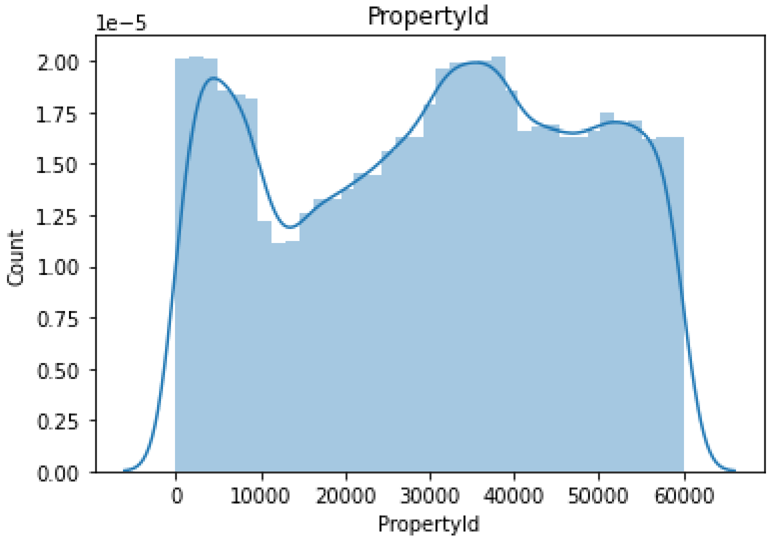
In [20]:
```
continuous_feature = [feature for feature in df1.columns if len(df1[feature].unique())>12]
print("Continuous feature Count :{}".format(len(continuous_feature)))
print(continuous_feature)
```

```
Continuous feature Count :8
['PropertyId', 'StoryCount', 'YearBuilt', 'UnitCount', 'NetRentableSF', 'GrossLandArea', 'OccupancyPercentage', 'YLT']
```

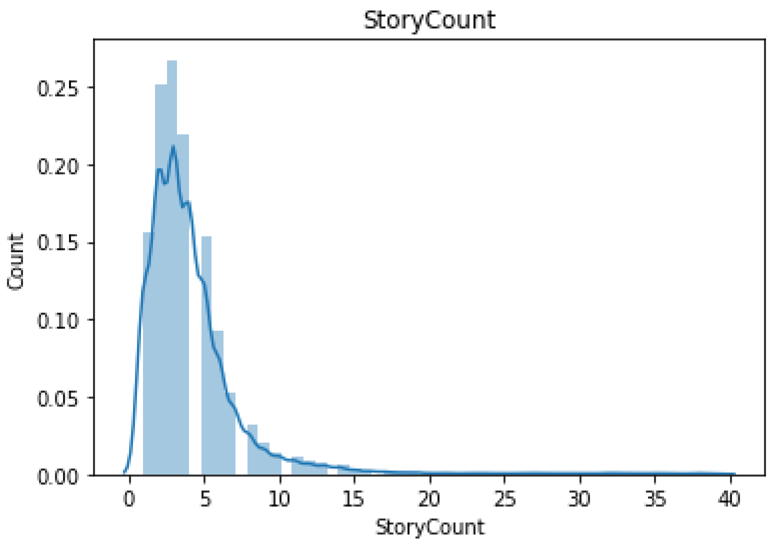
Plotting frequency distribution for continuous variables

In [21]:
```
for feature in continuous_feature:
    sns.distplot(df1[feature])
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

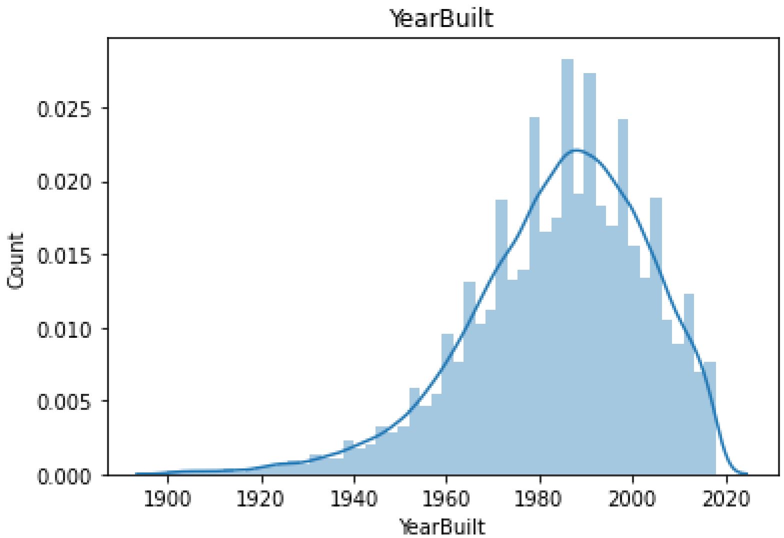
```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```



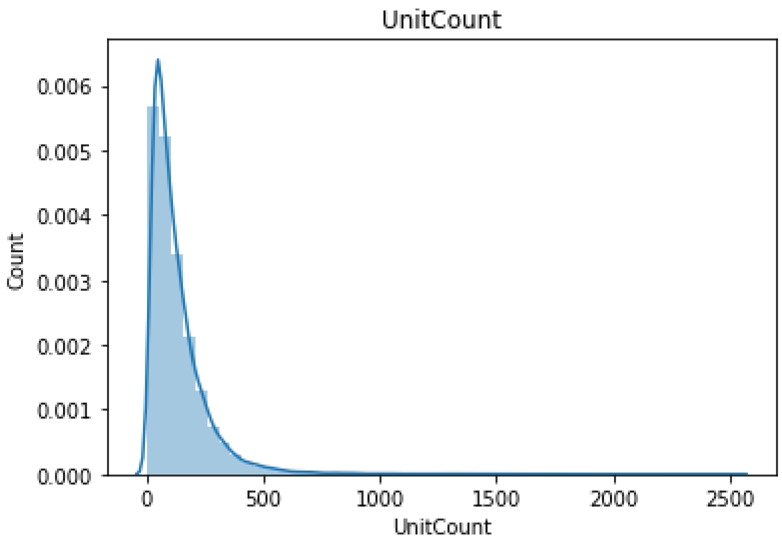
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).



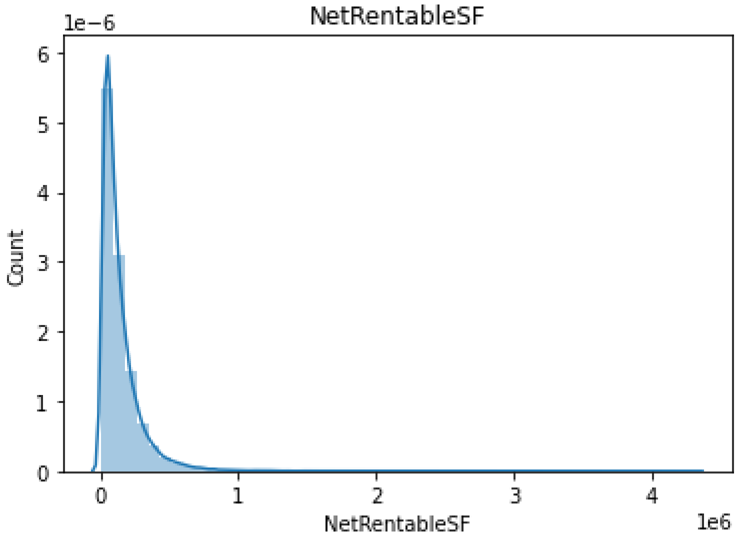
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).



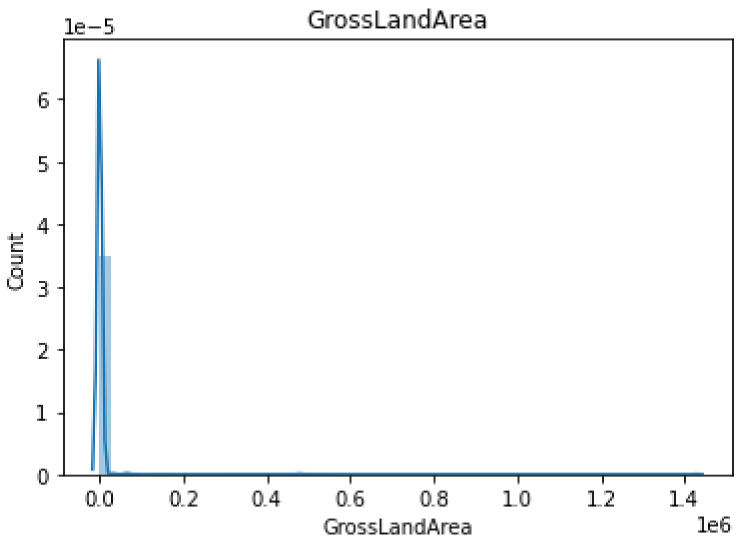
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).



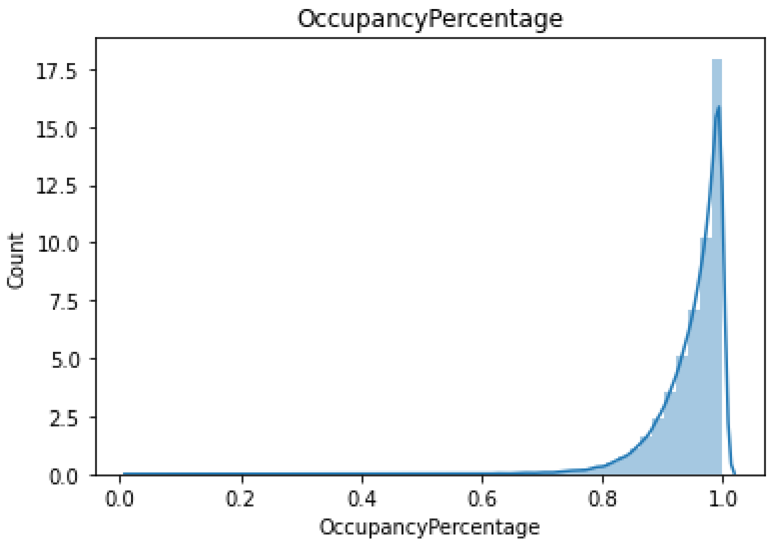
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).



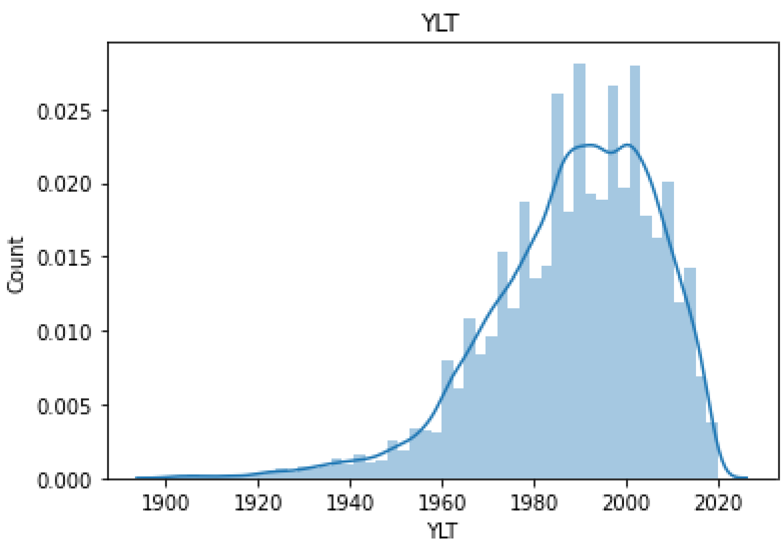

```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```




```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```

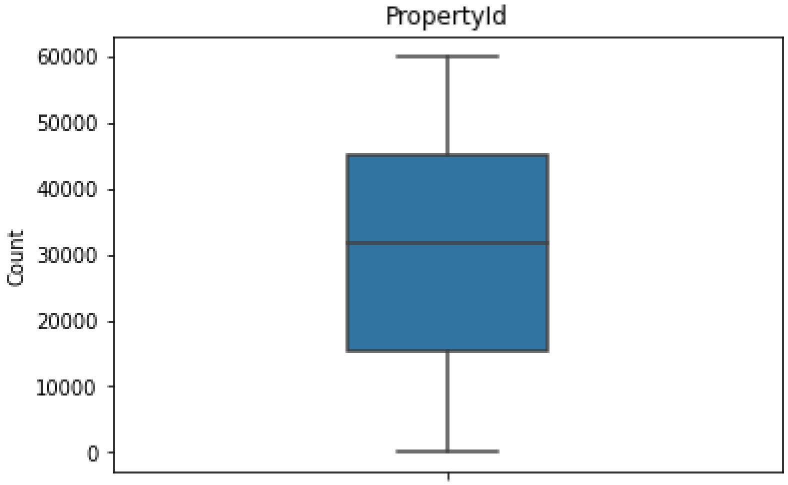


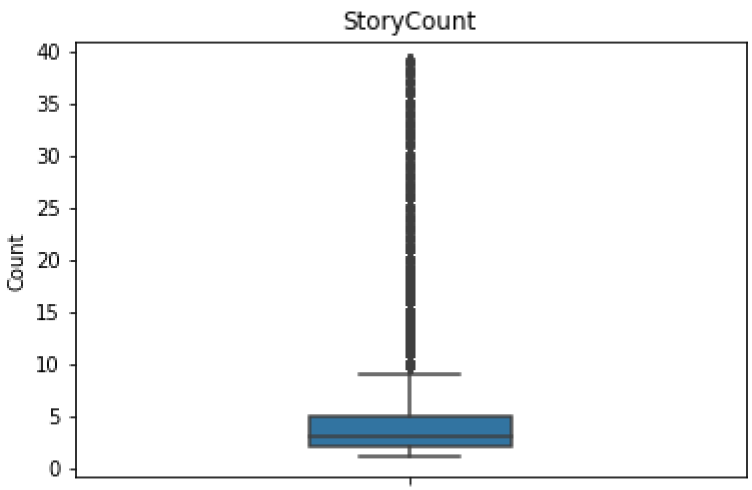

```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```

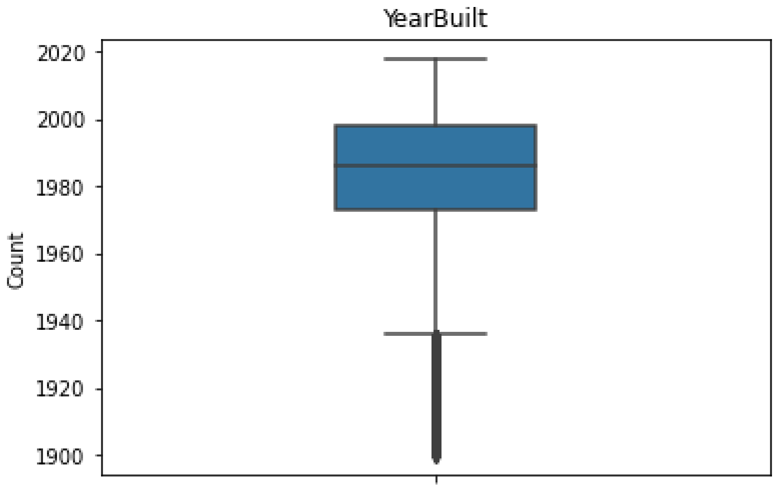


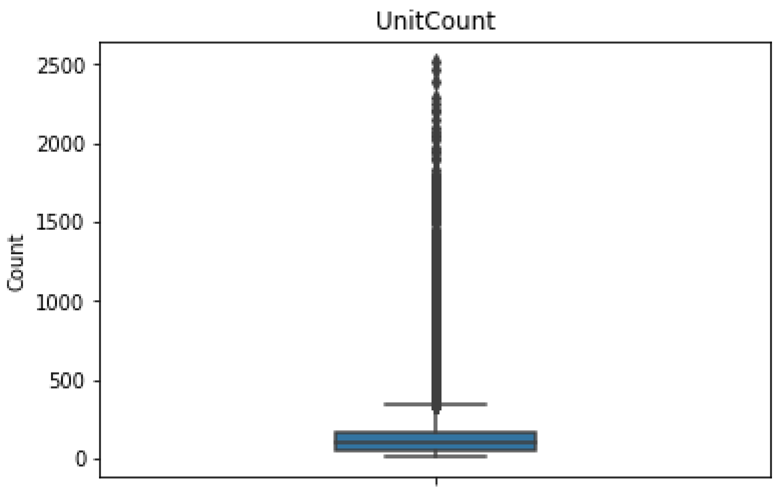
In [22]:

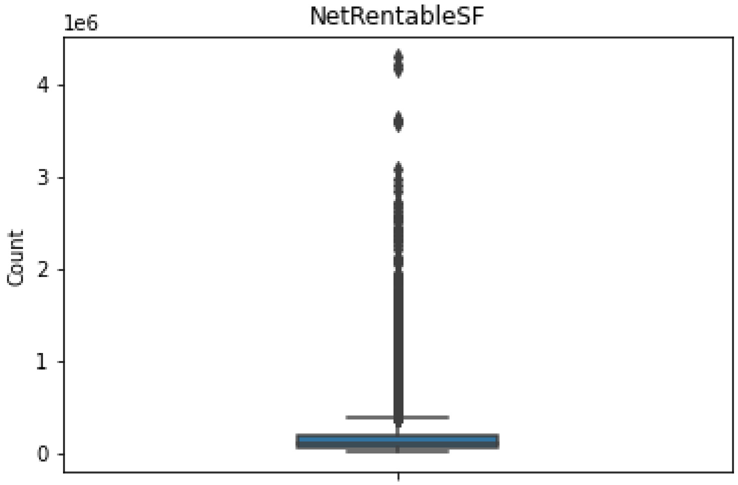
```
for feature in continuous_feature:
    sns.boxplot( y=df1[feature], width=0.3);
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

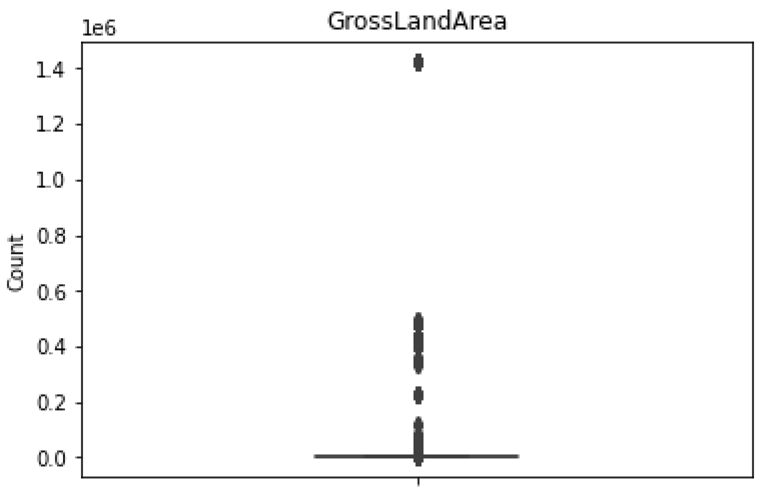

PropertyId
60000
50000
40000
30000
20000
10000
0
Count

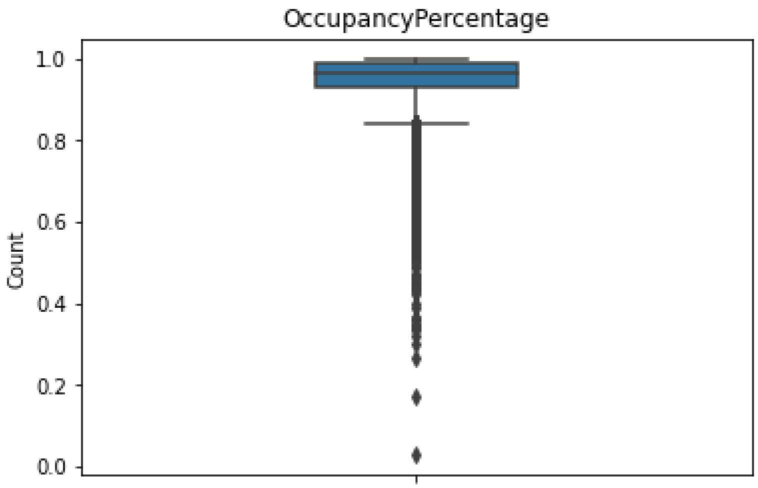

StoryCount
40
35
30
25
20
15
10
5
0
Count

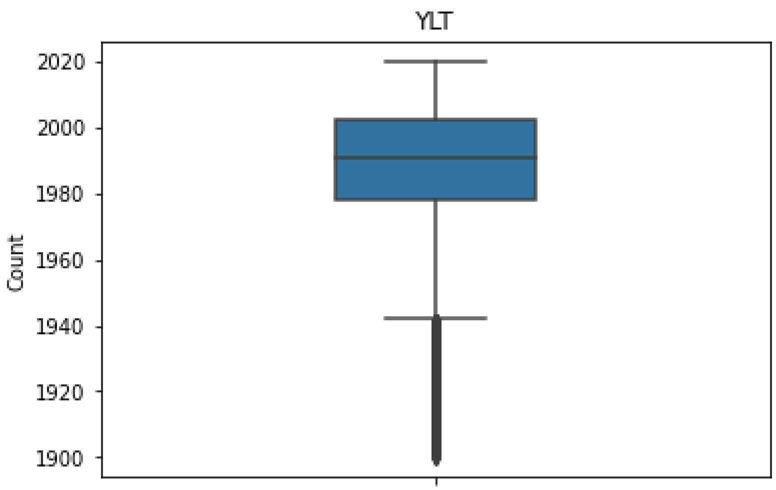

YearBuilt
Count
2020
2000
1980
1960
1940
1920
1900


UnitCount
Count
2500
2000
1500
1000
500
0


NetRentableSF
1e6
Count
4
3
2
1
0


GrossLandArea
1e6
Count
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0





In [107…

```
box_cox = ['YearBuilt','UnitCount','NetRentableSF','OccupancyPercentage','YLT']

for feature in box_cox:
   df1[feature],fitted_lambda = stats.boxcox(df1[feature])
```

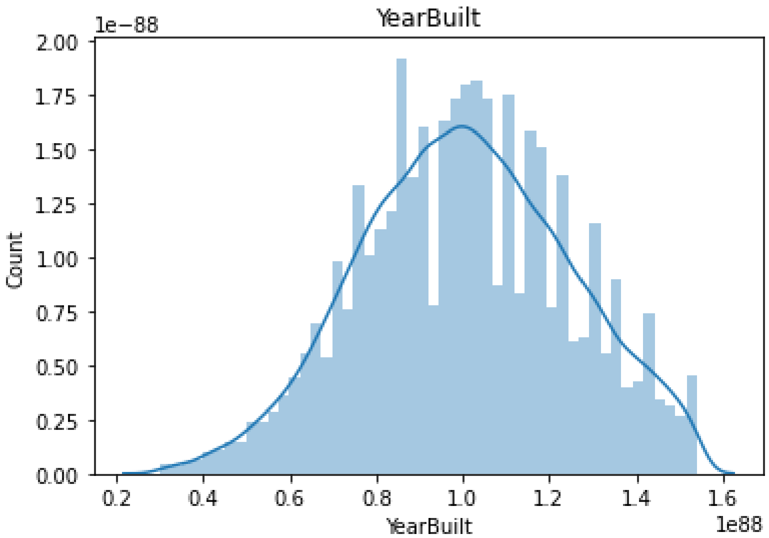
overflow encountered in multiply

In [109…

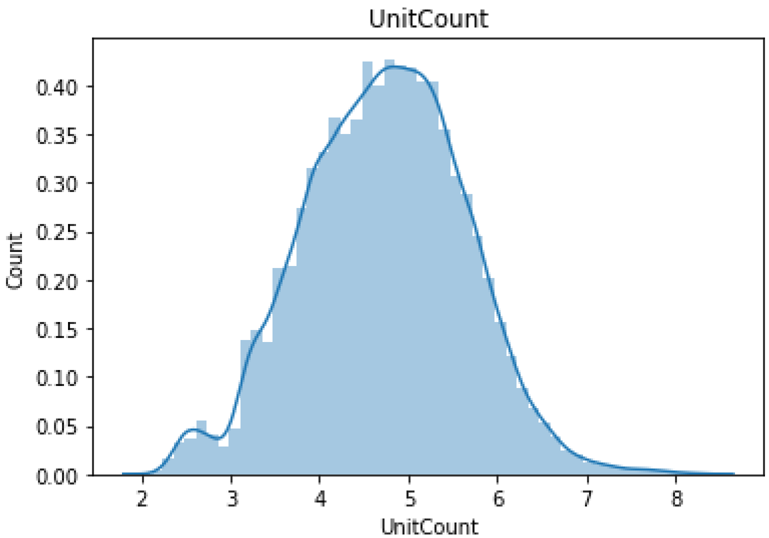
```
for feature in box_cox:
    sns.distplot(df1[feature])
    plt.ylabel("Count")
```

```
    plt.title(feature)
    plt.show()
```

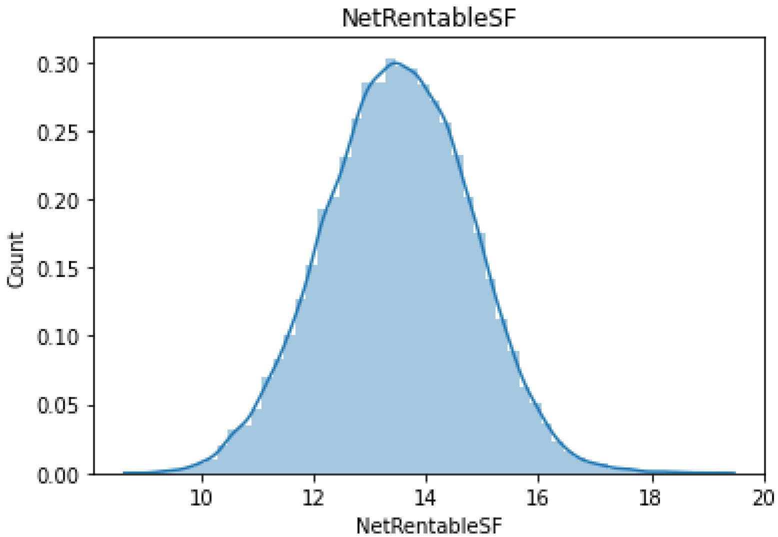
```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```



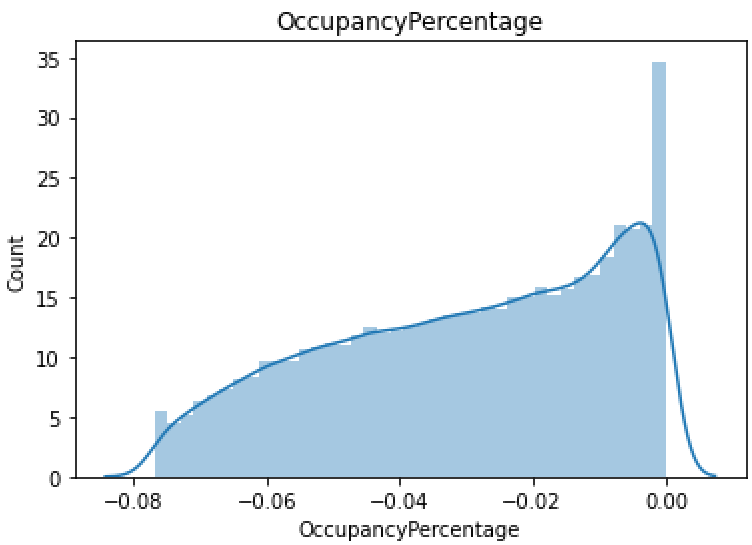

```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```



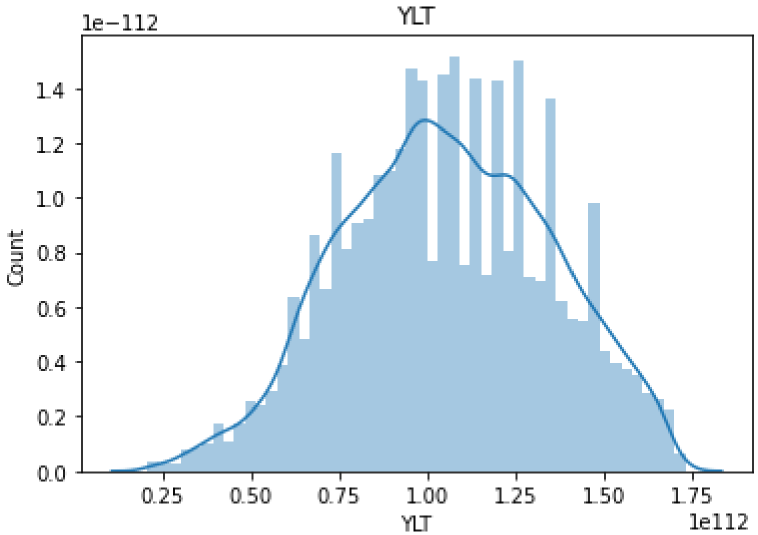
```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```




```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```



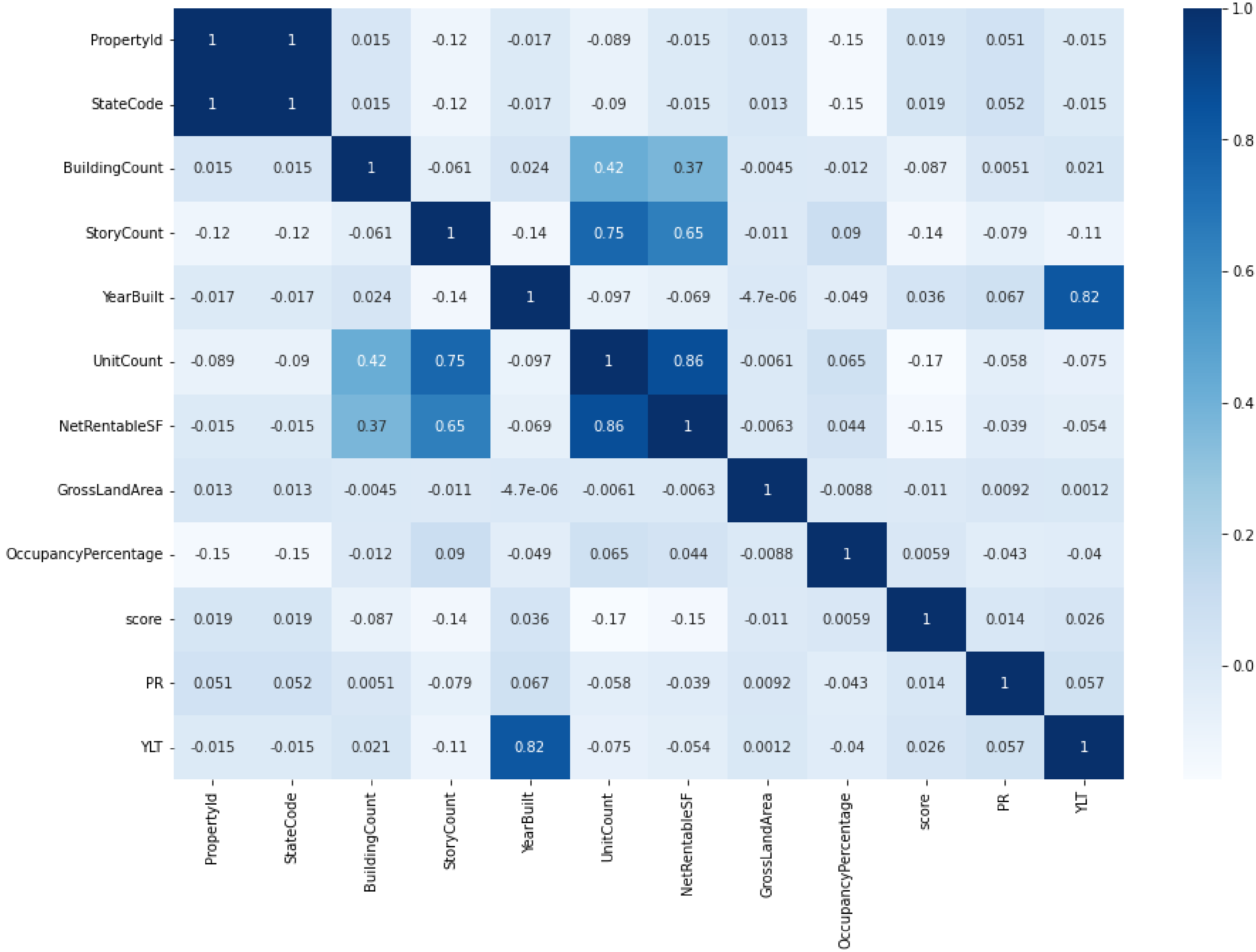

```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histo
grams).
```



Plotting correleation matrix to check correleation between features

In [23]:
```
plt.subplots(figsize=(15,10))
sns.heatmap(df1.corr(), cmap="Blues", annot=True)
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:

```
df1 = df1.drop(['PropertyId','NetRentableSF'],axis=1)
df1 = df1.reset_index(drop=True)
```

Dropping NetRentableSF since it is highly correlated with UnitCount

In [25]:
```
df1.columns = df1.columns.str.lower()
```

In [26]:
```
df_l = df1.copy()
```

In [27]:
```
df_census
```

Out[27]:

	STATECODE	CPIALL	MEDHHINC	MEDRENT	RECNO	MALMEDAGE	FEMMEDAGE	ANC_TOTALS	PRCNTSUN	SKYHOURS
0	0	218.705538	76709.12928	1131.010462	5	34.518548	36.894880	3.906696e+07	74	5
1	4	215.000249	64725.54587	856.409472	10	40.738115	43.656130	2.155436e+07	64	6
2	3	208.187286	67553.15527	668.116849	11	34.656151	37.137388	1.002270e+07	63	6
3	11	209.861941	58347.64959	520.353737	18	37.728493	40.432209	4.561936e+06	55	6
4	10	211.037230	61692.53362	624.214739	19	35.663126	38.471360	4.896363e+06	63	6
5	7	230.028723	93294.90389	1030.891483	31	37.929213	40.906936	8.463671e+06	56	6
6	2	225.174462	74339.07223	958.629564	33	36.820253	39.870720	1.862203e+07	51	7
7	6	211.167715	64183.29515	629.173158	34	36.923149	39.739212	1.072285e+07	60	6
8	9	206.768920	64859.93552	574.418972	36	38.147703	40.927839	1.120776e+07	51	7
9	8	211.184031	61430.02777	595.793176	43	37.830654	40.447696	7.170039e+06	59	6
10	1	203.123581	67713.64327	676.696093	44	33.426847	35.304262	2.788635e+07	66	5
11	5	219.923798	80154.11658	913.704306	48	37.416947	39.584375	9.103515e+06	48	7

In [28]:
```
df_census.columns = df_census.columns.str.lower()
```

In [29]:
```
df_r = df_census.copy()
```

In [30]:
```
df_r
```

Out[30]:

	statecode	cpiall	medhhinc	medrent	recno	malmedage	femmedage	anc_totals	prcntsun	skyhours	cleardays
0	0	218.705538	76709.12928	1131.010462	5	34.518548	36.894880	3.906696e+07	74	5	167
1	4	215.000249	64725.54587	856.409472	10	40.738115	43.656130	2.155436e+07	64	6	97
2	3	208.187286	67553.15527	668.116849	11	34.656151	37.137388	1.002270e+07	63	6	110
3	11	209.861941	58347.64959	520.353737	18	37.728493	40.432209	4.561936e+06	55	6	86
4	10	211.037230	61692.53362	624.214739	19	35.663126	38.471360	4.896363e+06	63	6	105
5	7	230.028723	93294.90389	1030.891483	31	37.929213	40.906936	8.463671e+06	56	6	94
6	2	225.174462	74339.07223	958.629564	33	36.820253	39.870720	1.862203e+07	51	7	65
7	6	211.167715	64183.29515	629.173158	34	36.923149	39.739212	1.072285e+07	60	6	108
8	9	206.768920	64859.93552	574.418972	36	38.147703	40.927839	1.120776e+07	51	7	73
9	8	211.184031	61430.02777	595.793176	43	37.830654	40.447696	7.170039e+06	59	6	109
10	1	203.123581	67713.64327	676.696093	44	33.426847	35.304262	2.788635e+07	66	5	140
11	5	219.923798	80154.11658	913.704306	48	37.416947	39.584375	9.103515e+06	48	7	79

Performing left join to merge the main dataframe with the census dataframe

In [31]:

```
df_main = df_l.merge(df_r, on='statecode', how='left')
```

In [32]:

```
df_main
```

Out[32]:

	statecode	buildingcount	storycount	yearbuilt	unitcount	grosslandarea	propertytype	propertysubtype	occupancypercentag
0	0	2	3	1999	90	16.00	Multifamily	Other	0.777628
1	0	2	4	1991	102	13.39	Multifamily	Other	0.97622
2	0	2	1	1993	22	2.09	Multifamily	Other	0.938526
3	0	2	4	1973	124	0.66	Multifamily	Other	0.99561
4	0	2	4	2006	92	4.30	Multifamily	Other	0.99962
...	...	...	...	...	...	...	...	...	..
48014	11	2	4	1995	125	38.00	Multifamily	Other	0.92973

	statecode	buildingcount	storycount	yearbuilt	unitcount	grosslandarea	propertytype	propertysubtype	occupancypercentage
48015	11	3	1	1969	51	0.00	Multifamily	Other	0.89680
48016	11	2	2	1997	60	9.00	Multifamily	Other	0.86022
48017	11	5	5	1965	320	0.00	Multifamily	Other	0.85333
48018	11	4	23	1980	1031	0.00	Multifamily	Other	0.99262

48019 rows × 26 columns

In [33]:

```
columns = df_main.columns
print(columns)
```

```
Index(['statecode', 'buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea', 'propertytype', 'propertysubtype',
       'occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow'],
      dtype='object')
```

Encoding Statecode with a propability score

In [34]:

```
state_enc = df_main.groupby(['statecode'])['score'].agg('mean').reset_index()
state_enc = state_enc.rename(columns = {'score': 'statecodeenc'})
```

In [35]:

```
new_data = df_main.merge(state_enc, on = 'statecode', how = 'inner')
```

In [36]:

```
new_data.columns
```

Out[36]:

```
Index(['statecode', 'buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea', 'propertytype', 'propertysubtype',
       'occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow', 'statecodeenc'],
      dtype='object')
```

In [37]:
```
new_data.drop(['statecode'], axis = 'columns', inplace = True)
```

Dropping feature statecode since it has been encoded into statecodeenc

In [38]:
```
new_data.shape
```

Out[38]:
```
(48019, 26)
```

In [39]:
```
new_data.columns
```

Out[39]:
```
Index(['buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea', 'propertytype', 'propertysubtype',
       'occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow', 'statecodeenc'],
      dtype='object')
```

Splitting dataset

In [40]:
```
x = new_data.drop(['score'],axis = 1)
y = new_data['score']
```

In [41]:
```
x_dev, x_test, y_dev, y_test = train_test_split(
    x, y, test_size=0.2, random_state = 100)

x_train, x_val, y_train, y_val = train_test_split(
    x_dev, y_dev, test_size=0.25, random_state = 100)
```

In [42]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28811 entries, 7893 to 14260
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   buildingcount        28811 non-null  int64
 1   storycount           28811 non-null  int64
 2   yearbuilt            28811 non-null  int64
```

```
 3   unitcount             28811 non-null  int64  
 4   grosslandarea         28811 non-null  float64
 5   propertytype          28811 non-null  object 
 6   propertysubtype       28811 non-null  object 
 7   occupancypercentage   28811 non-null  float64
 8   pr                    28811 non-null  int64  
 9   ylt                   28811 non-null  int64  
 10  cpiall                28811 non-null  float64
 11  medhhinc              28811 non-null  float64
 12  medrent               28811 non-null  float64
 13  recno                 28811 non-null  int64  
 14  malmedage             28811 non-null  float64
 15  femmedage             28811 non-null  float64
 16  anc_totals            28811 non-null  float64
 17  prcntsun              28811 non-null  int64  
 18  skyhours              28811 non-null  int64  
 19  cleardays             28811 non-null  int64  
 20  raindays              28811 non-null  int64  
 21  snowdays              28811 non-null  int64  
 22  annulrain             28811 non-null  int64  
 23  annulsnow             28811 non-null  int64  
 24  statecodeenc          28811 non-null  float64
dtypes: float64(9), int64(14), object(2)
memory usage: 5.7+ MB
```

Performing scaling on numerical features and encoding on categorical features

In [43]:

```
numerical = ['buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea', 'occupancypercentage', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow', 'statecodeenc']


categorical = ['propertytype', 'propertysubtype']

numerical_transformer = make_pipeline(RobustScaler())
categorical_transformer = make_pipeline(OneHotEncoder())

preprocess = make_column_transformer((numerical_transformer, numerical),(categorical_transformer, categorical),

x_train = preprocess.fit_transform(x_train, y = y_train)
x_val = preprocess.transform(x_val)
x_test = preprocess.transform(x_test)
```

In [44]:
```
num_feature_list = ['buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea','occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow', 'statecodeenc']
cat_feature_list = preprocess.transformers_[1][1]['onehotencoder'].get_feature_names()
feature_list = num_feature_list.extend(cat_feature_list)
```

```
Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Pl
ease use get_feature_names_out instead.
```

Applying SMOTE for upsampling minority class

In [45]:
```
from imblearn.over_sampling import SMOTE
```

In [46]:
```
os = SMOTE(random_state=13)
x_train_os, y_train_os = os.fit_resample(x_train, y_train)
```

In [47]:
```
print('Original dataset shape {}'.format(Counter(y_train)))
print('Resampled dataset shape {}'.format(Counter(y_train_os)))
```

```
Original dataset shape Counter({1: 21170, 0: 7641})
Resampled dataset shape Counter({0: 21170, 1: 21170})
```

Model building -1 (Logistic regression)

In [54]:
```
lr = LogisticRegression()
lr.fit(x_train_os,y_train_os) #Note that regularization is applied by default
```

```
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Out[54]:

```
▾ LogisticRegression
LogisticRegression()
```

Evaluating model performance

In [55]:

```
lr_y_pred = lr.predict(x_test)
lr_acc = accuracy_score(y_test, lr_y_pred)
print('Accuracy of logistic regression:',lr_acc)
precision = precision_score(y_test, lr_y_pred, average='weighted')
print("Precision of logistic regression:", precision)
recall = recall_score(y_test, lr_y_pred)
print("Recall of logistic regression:", recall)
F1score = f1_score(y_test, lr_y_pred)
print("F1 score of logistic regression:", F1score)
lr_cm = confusion_matrix(y_test, lr_y_pred)
print('Confusion matrix for Logistic Regression:\n',lr_cm)
```

```
Accuracy of logistic regression: 0.6437942523948355
Precision of logistic regression: 0.7303748979672338
Recall of logistic regression: 0.6260277856535299
F1 score of logistic regression: 0.7208030686362523
Confusion matrix for Logistic Regression:
 [[1767  783]
 [2638 4416]]
```

In [56]:

```
lr_y_pred_proba = lr.predict_proba(x_test)[::,1]
#calculate AUC of model
auc = metrics.roc_auc_score(y_test, lr_y_pred_proba)
#print AUC score
print(auc)
```

```
0.705506095832152
```

Plotting feature importance for logistic regression

In [138…

```
num_feature_list = ['buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea','occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', 'snowdays', 'annulrain',
       'annulsnow', 'statecodeenc']
```

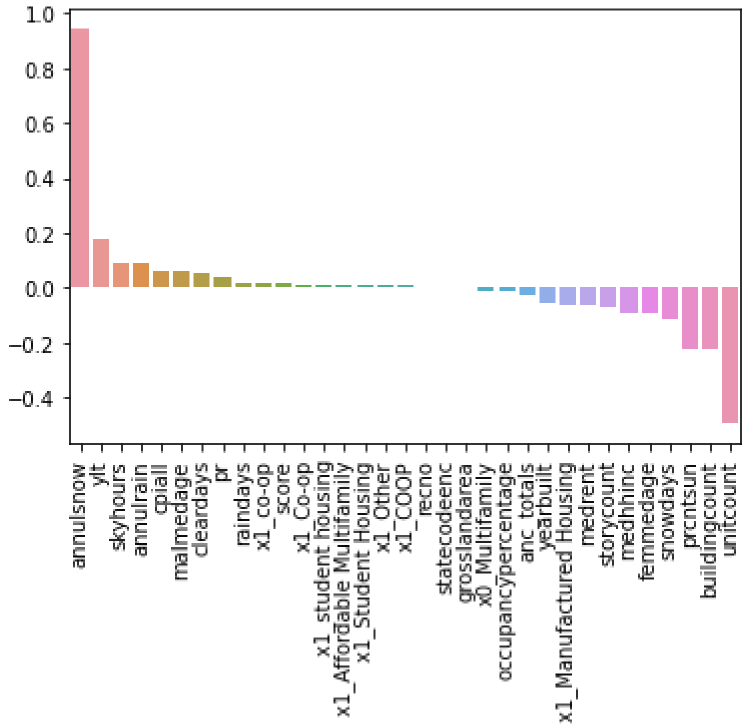
```
cat_feature_list = preprocess.transformers_[1][1]['onehotencoder'].get_feature_names()
num_feature_list.extend(cat_feature_list)
```

```
Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Pl
ease use get_feature_names_out instead.
```

In [139…

```
a = list(zip(num_feature_list, lr.coef_[0]))
features, imps = zip(*(sorted(list(filter(lambda x: x[1] != 0, a)), key = lambda x: x[1], reverse = True)))

ax = sns.barplot(x = list(features), y = list(imps))
ax.tick_params(axis = 'x', rotation = 90)
```



Model building -2 (Random Forest Classifier)

Performing randomized search CV to obtain the best set of hyperparameters for our model

In [ ]:

```
pipe_rfc = RandomForestClassifier()

params = {'bootstrap': [True, False],
```

```
 'max_depth': [10, 20, 30, 50,70, 80, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10,30,50,100,200, 500]}

search = RandomizedSearchCV(pipe_rfc, params, cv = 5)
search.fit(x_train_os, y_train_os)
print(search.best_params_)
```

In [51]:

```
pipe_rfc = RandomForestClassifier()
pipe_rfc.fit(x_train_os, y_train_os)
rfc_y_pred_test =  pipe_rfc.predict(x_test)
rfc_y_pred_val = pipe_rfc.predict(x_val)
```

Evaluating model performance

In [52]:

```
rfc_acc = accuracy_score(y_test, rfc_y_pred_test)
print('Accuracy of Random forest classifier:',rfc_acc)
precision = precision_score(y_test, rfc_y_pred_test, average='weighted')
print("Precision of Random forest classifier:", precision)
recall = recall_score(y_test, rfc_y_pred_test)
print("Recall of Random forest classifier:", recall)
F1score = f1_score(y_test, rfc_y_pred_test)
print("F1 score of Random forest classifier:", F1score)
rfc_cm = confusion_matrix(y_test, rfc_y_pred_test)
print('Confusion matrix for :\n',rfc_cm)
```

```
Accuracy of Random forest classifier: 0.7091836734693877
Precision of Random forest classifier: 0.7030239918829337
Recall of Random forest classifier: 0.8141480011341083
F1 score of Random forest classifier: 0.804398067091533
Confusion matrix for :
 [[1068 1482]
 [1311 5743]]
```

In [53]:

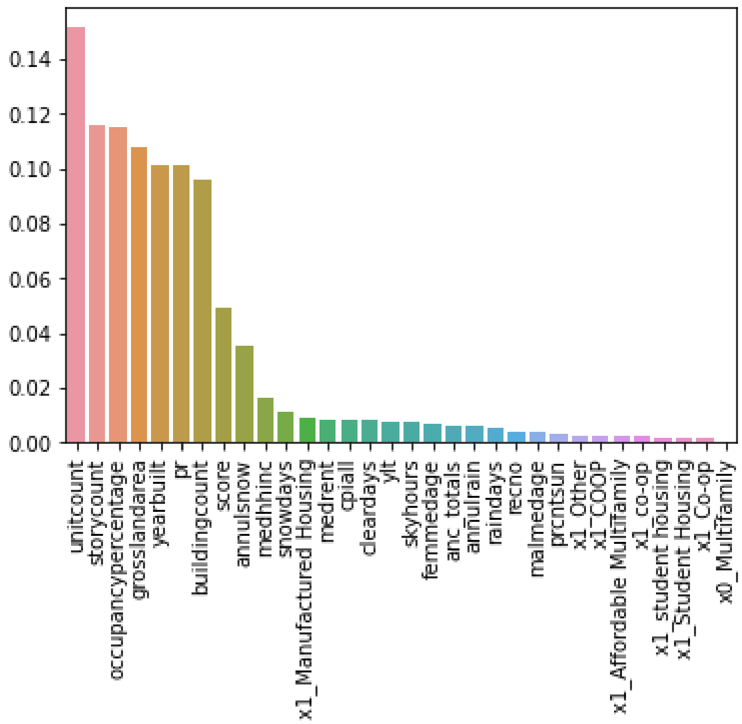
```
rfc_y_pred_proba = pipe_rfc.predict_proba(x_test)[::,1]
#calculate AUC of model
auc = metrics.roc_auc_score(y_test, rfc_y_pred_proba)
#print AUC score
print(auc)
```

```
0.7035198774718281
```

Plotting feature importance for Random forest algorithm

In [143…

```
b = list(zip(num_feature_list, pipe_rfc.feature_importances_))
features, imps = zip(*(sorted(list(filter(lambda x: x[1] != 0, b)), key = lambda x: x[1], reverse = True)))
bx = sns.barplot(x = list(features), y = list(imps))
bx.tick_params(axis = 'x', rotation = 90)
```



In [ ]:

```
import fasttreeshap
explainer = fasttreeshap.TreeExplainer(pipe_rfc, algorithm = 'auto', n_jobs = 1)
shap_values = explainer.shap_values(x_test).values
shap_values
```

Model building -3 (XGB Classifier)

In [62]:

```
xgb = XGBClassifier()
```

```
params_xgb = {'max_depth': [3, 6, 10, 15,20],
              'learning_rate': [0.01, 0.1, 0.2, 0.3, 0.4],
              'subsample': np.arange(0.5, 1.0, 0.1),
              'colsample_bytree': np.arange(0.5, 1.0, 0.1),
              'colsample_bylevel': np.arange(0.5, 1.0, 0.1),
              'n_estimators': [100, 250, 500, 750],}
```

In [ ]:

```
search_xgb = RandomizedSearchCV(xgb, params_xgb, cv = 5)
search.fit(x_train_os, y_train_os)
print(search.best_params_)
```

In [48]:

```
pipe_xgb = XGBClassifier(n_estimators=500, min_samples_split=3, min_samples_leaf=2, max_features='auto', max_de

pipe_xgb.fit(x_train_os, y_train_os)
xgb_y_pred_test =  pipe_xgb.predict(x_test)
xgb_y_pred_val = pipe_xgb.predict(x_val)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this
warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and
2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
[17:29:33] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with
the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like
to restore the old behavior.
```

Evaluating model performance

In [49]:

```
xgb_acc = accuracy_score(y_test, xgb_y_pred_test)
print('Accuracy of XG Boost classifier:',xgb_acc)
precision = precision_score(y_test, xgb_y_pred_test, average='weighted')
print("Precision of XG Boost classifier:", precision)
recall = recall_score(y_test, xgb_y_pred_test)
print("Recall of XG Boost classifier:", recall)
F1score = f1_score(y_test, xgb_y_pred_test)
print("F1 score of XG Boost classifier:", F1score)
rfc_cm = confusion_matrix(y_test, xgb_y_pred_test)
print('Confusion matrix for :\n',rfc_cm)
```

```
Accuracy of XG Boost classifier: 0.7312578092461475
Precision of XG Boost classifier: 0.7061022392364593
Recall of XG Boost classifier: 0.8755316132690671
```

```
F1 score of XG Boost classifier: 0.8271613205651912
Confusion matrix for :
 [[ 847 1703]
 [ 878 6176]]
```

In [50]:

```
xgb_y_pred_proba = pipe_xgb.predict_proba(x_test)[::,1]
#calculate AUC of model
auc = metrics.roc_auc_score(y_test, xgb_y_pred_proba)
#print AUC score
print(auc)
```

```
0.7139276283237991
```

Plotting feature importance for XGB Classifier

In [147…

```
c = list(zip(num_feature_list, pipe_xgb.feature_importances_))
features, imps = zip(*(sorted(list(filter(lambda x: x[1] != 0, c)), key = lambda x: x[1], reverse = True)))
bx = sns.barplot(x = list(features), y = list(imps))
bx.tick_params(axis = 'x', rotation = 90)
```