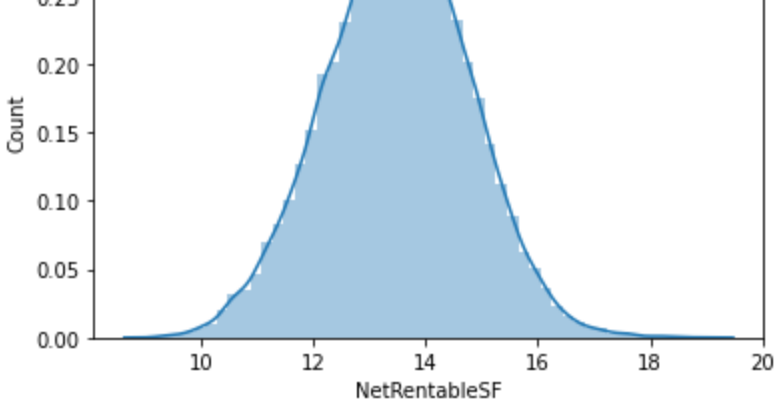
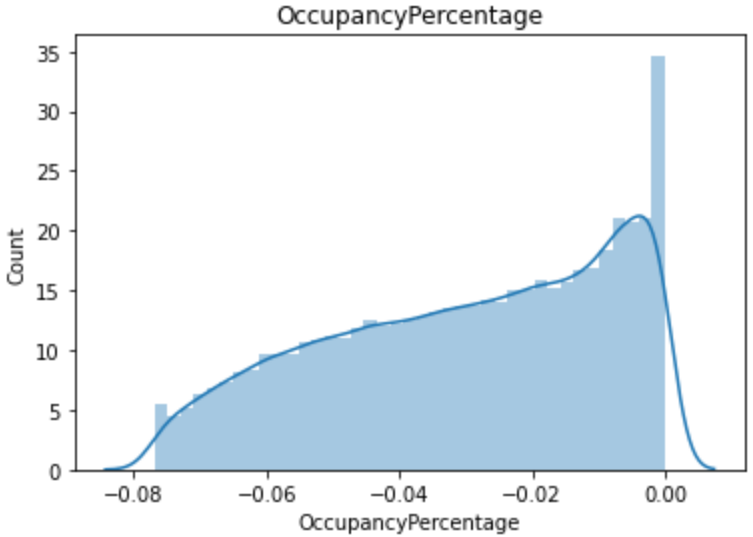
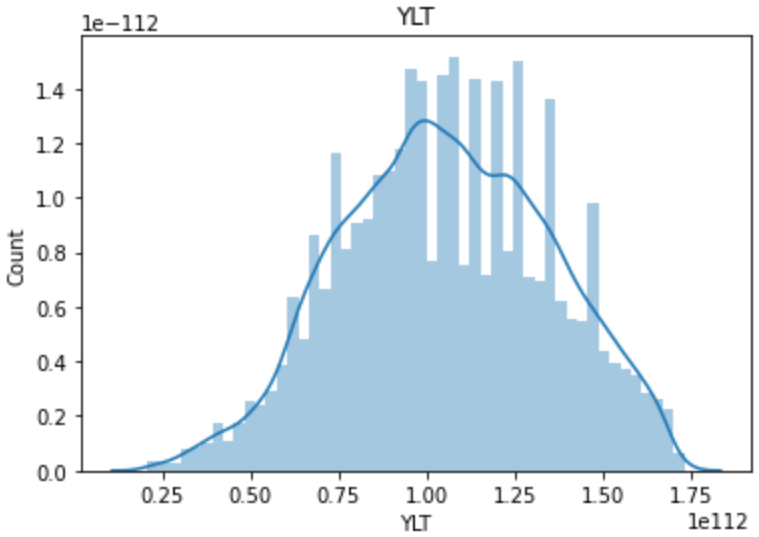


`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for his
tograms).



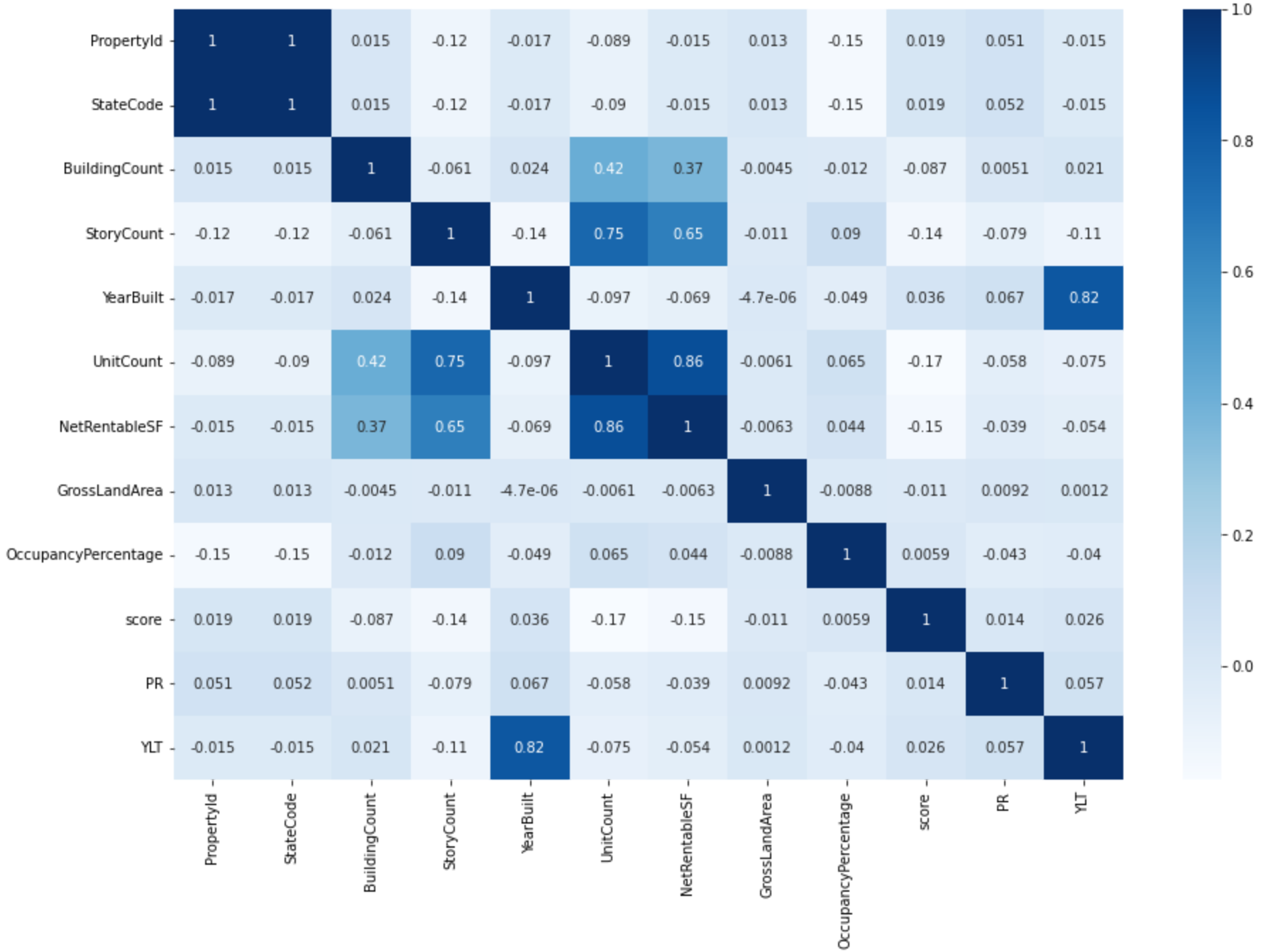
```
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for his
tograms).
```



Plotting correleation matrix to check correleation between features

In [21]:
```
plt.subplots(figsize=(15,10))
sns.heatmap(df1.corr(), cmap="Blues", annot=True)
```

Out[21]: <AxesSubplot:>


PropertyId 1 1 0.015 -0.12 -0.017 -0.089 -0.015 0.013 -0.15 0.019 0.051 -0.015
StateCode 1 1 0.015 -0.12 -0.017 -0.09 -0.015 0.013 -0.15 0.019 0.052 -0.015
BuildingCount 0.015 0.015 1 -0.061 0.024 0.42 0.37 -0.0045 -0.012 -0.087 0.0051 0.021
StoryCount -0.12 -0.12 -0.061 1 -0.14 0.75 0.65 -0.011 0.09 -0.14 -0.079 -0.11
YearBuilt -0.017 -0.017 0.024 -0.14 1 -0.097 -0.069 -4.7e-06 -0.049 0.036 0.067 0.82
UnitCount -0.089 -0.09 0.42 0.75 -0.097 1 0.86 -0.0061 0.065 -0.17 -0.058 -0.075
NetRentableSF -0.015 -0.015 0.37 0.65 -0.069 0.86 1 -0.0063 0.044 -0.15 -0.039 -0.054
GrossLandArea 0.013 0.013 -0.0045 -0.011 -4.7e-06 -0.0061 -0.0063 1 -0.0088 -0.011 0.0092 0.0012
OccupancyPercentage -0.15 -0.15 -0.012 0.09 -0.049 0.065 0.044 -0.0088 1 0.0059 -0.043 -0.04
score 0.019 0.019 -0.087 -0.14 0.036 -0.17 -0.15 -0.011 0.0059 1 0.014 0.026
PR 0.051 0.052 0.0051 -0.079 0.067 -0.058 -0.039 0.0092 -0.043 0.014 1 0.057
YLT -0.015 -0.015 0.021 -0.11 0.82 -0.075 -0.054 0.0012 -0.04 0.026 0.057 1
PropertyId
StateCode
BuildingCount
StoryCount
YearBuilt
UnitCount
NetRentableSF
GrossLandArea
OccupancyPercentage
score
PR
YLT
1.0
0.8
0.6
0.4
0.2
0.0

In [22]:
```
df1 = df1.drop(['PropertyId','NetRentableSF'],axis=1)
df1 = df1.reset_index(drop=True)
```

Dropping NetRentableSF since it is highly correlated with UnitCount

In [23]:
```
df1.columns = df1.columns.str.lower()
```

In [24]:
```
df_l = df1.copy()
```

In [25]:
```
df_census
```

Out[25]:

	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

In [26]:
```
df_census.columns = df_census.columns.str.lower()
```

In [27]:
```
df_r = df_census.copy()
```

In [28]:
```
df_r
```

Out[28]:

	statecode	cpiall	medhhinc	medrent	recno	malmedage	femmedage	anc_totals	prcntsun	skyhours	cleardays	raindays	snowd
0	0	218.705538	76709.12928	1131.010462	5	34.518548	36.894880	3.906696e+07	74	5	167	55	
1	4	215.000249	64725.54587	856.409472	10	40.738115	43.656130	2.155436e+07	64	6	97	115	
2	3	208.187286	67553.15527	668.116849	11	34.656151	37.137388	1.002270e+07	63	6	110	111	
3	11	209.861941	58347.64959	520.353737	18	37.728493	40.432209	4.561936e+06	55	6	86	130	
4	10	211.037230	61692.53362	624.214739	19	35.663126	38.471360	4.896363e+06	63	6	105	104	
5	7	230.028723	93294.90389	1030.891483	31	37.929213	40.906936	8.463671e+06	56	6	94	116	
6	2	225.174462	74339.07223	958.629564	33	36.820253	39.870720	1.862203e+07	51	7	65	150	
7	6	211.167715	64183.29515	629.173158	34	36.923149	39.739212	1.072285e+07	60	6	108	117	
8	9	206.768920	64859.93552	574.418972	36	38.147703	40.927839	1.120776e+07	51	7	73	141	
9	8	211.184031	61430.02777	595.793176	43	37.830654	40.447696	7.170039e+06	59	6	109	118	
10	1	203.123581	67713.64327	676.696093	44	33.426847	35.304262	2.788635e+07	66	5	140	72	
11	5	219.923798	80154.11658	913.704306	48	37.416947	39.584375	9.103515e+06	48	7	79	128	

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

In [29]:
```
df_main = df_l.merge(df_r, on='statecode', how='left')
```

In [30]:
```
df_main
```

Out[30]:

	statecode	buildingcount	storycount	yearbuilt	unitcount	grosslandarea	propertytype	propertysubtype	occupancypercentage	score	...	mal

	statecode	buildingcount	storycount	yearbuilt	unitcount	grosslandarea	propertytype	propertysubtype	occupancypercentage	score	...	mal
0	0	2	3	1999	90	16.00	Multifamily	Other	0.777628	1	...	34
1	0	2	4	1991	102	13.39	Multifamily	Other	0.976221	1	...	34
2	0	2	1	1993	22	2.09	Multifamily	Other	0.938526	0	...	34
3	0	2	4	1973	124	0.66	Multifamily	Other	0.995617	1	...	34
4	0	2	4	2006	92	4.30	Multifamily	Other	0.999629	1	...	34
...	...	...	...	...	...	...	...	...	...	...	...	
48014	11	2	4	1995	125	38.00	Multifamily	Other	0.929732	1	...	37
48015	11	3	1	1969	51	0.00	Multifamily	Other	0.896801	1	...	37
48016	11	2	2	1997	60	9.00	Multifamily	Other	0.860220	1	...	37
48017	11	5	5	1965	320	0.00	Multifamily	Other	0.853336	0	...	37
48018	11	4	23	1980	1031	0.00	Multifamily	Other	0.992629	0	...	37

48019 rows × 26 columns

In [31]:
```
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 [32]:
```
state_enc = df_main.groupby(['statecode'])['score'].agg('mean').reset_index()
state_enc = state_enc.rename(columns = {'score': 'statecodeenc'})
```

In [33]:
```
new_data = df_main.merge(state_enc, on = 'statecode', how = 'inner')
```

In [34]:
```
new_data.columns
```

Out[34]:
```
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 [35]:
```
new_data.drop(['statecode'], axis = 'columns', inplace = True)
```

Dropping feature statecode since it has been encoded into statecodeenc

In [36]:
```
new_data.shape
```

Out[36]:
```
(48019, 26)
```

In [37]:
```
new_data.columns
```

Out[37]:
```
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 [38]:
```
x = new_data.drop(['score'],axis = 1)
y = new_data['score']
```

In [39]:
```
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 [40]:
```
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 [41]:
```
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 [42]:
```
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. P
lease use get_feature_names_out instead.
```

Applying SMOTE for upsampling minority class

In [43]:
```
from imblearn.over_sampling import SMOTE
```

In [44]:
```
os = SMOTE(random_state=13)
x_train_os, y_train_os = os.fit_resample(x_train, y_train)
```

In [45]:
```
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 [46]:
```
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
```

Out[46]:

```
▾ LogisticRegression

LogisticRegression()
```

Evaluating model performance

In [47]:

```
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 [48]:

```
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 score
print(auc)
```

```
0.705506095832152
```

Plotting feature importance for logistic regression

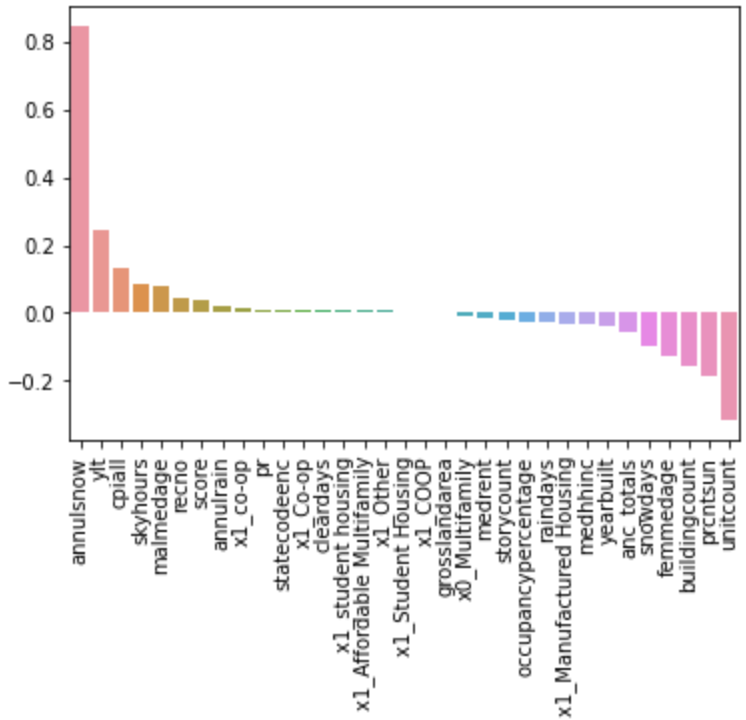
In [49]:
```
num_feature_list = ['buildingcount', 'storycount', 'yearbuilt', 'unitcount',
       'grosslandarea','occupancypercentage', 'score', 'pr', 'ylt', 'cpiall', 'medhhinc',
       'medrent', 'recno', 'malmedage', 'femmedage', 'anc_totals', 'prcntsun',
       'skyhours', 'cleardays', 'raindays', '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. P
lease use get_feature_names_out instead.
```

In [50]:
```
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 = sns.barplot(x = list(features), y = list(imps))
ax.tick_params(axis = 'x', rotation = 90)
```



Model building -2 (Random Forest Classifier)

In [ ]:
```
pipe_rfc = RandomForestClassifier()

params = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 50, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10,30,50,100,200]}

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

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

In [51]:
```
lomForestClassifier(bootstrap = True,max_depth = 20,min_samples_leaf = 1,min_samples_split = 2,n_estimators = 50)
_train_os, y_train_os)
: =  pipe_rfc.predict(x_test)
```

```
= 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: 0.7109537692628072
Precision of Random forest classifier: 0.7037412380517062
Recall of Random forest classifier: 0.8175503260561383
F1 score of Random forest classifier: 0.8060097833682739
Confusion matrix for :
 [[1061 1489]
 [1287 5767]]
```

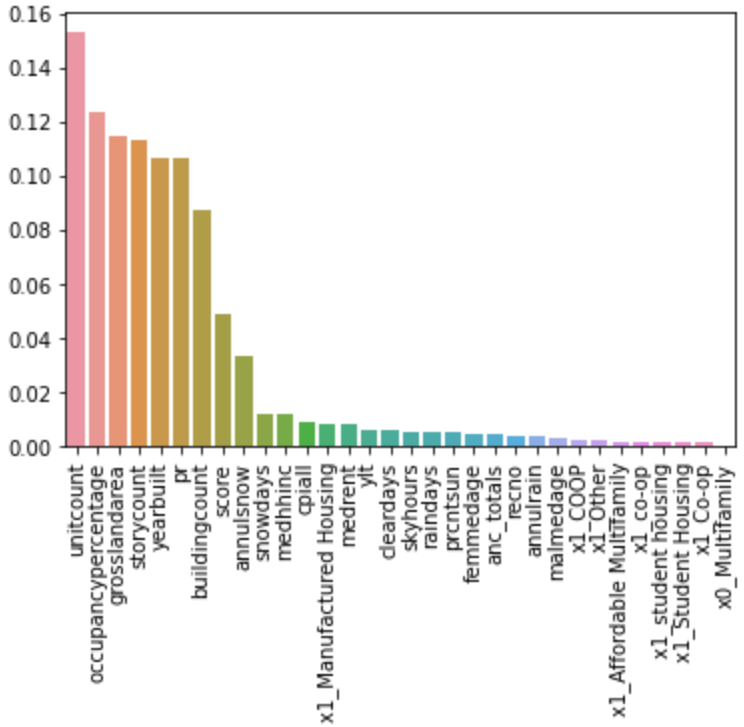
In [53]:
```
rfc_y_pred_proba = pipe_rfc.predict_proba(x_test)[::,1]
#calculate AUC of model
auc = metrics.roc_auc_score(y_test, rfc_y_pred_proba)
#print AUC score
print(auc)
```

```
0.7032909432556692
```

Plotting feature importance for Random forest algorithm

In [54]:
```
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)
```



Model building -3 (XGB Classifier)

In [55]:
```
xgb = XGBClassifier()

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

In [56]:
```
.er(n_estimators=500, min_samples_split=3, min_samples_leaf=2, max_features='auto', max_depth=20, bootstrap=True)
)s, y_train_os)
)e_xgb.predict(x_test)
_xgb.predict(x_val)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove thi
s warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; an
d 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[18:14:47] 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 lik
e to restore the old behavior.
```

Evaluating model performance

In [57]:
```
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]]
```

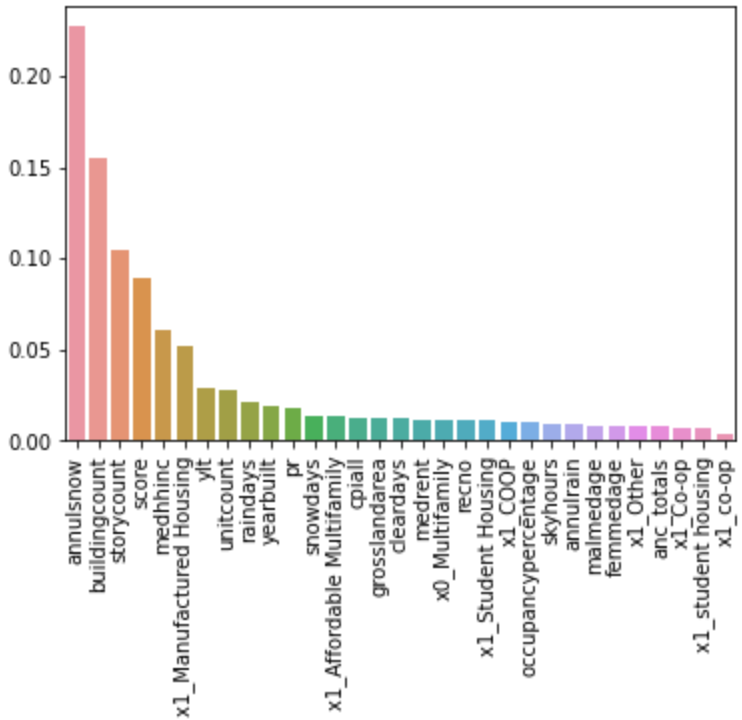
In [58]:
```
xgb_y_pred_proba = pipe_xgb.predict_proba(x_test)[::,1]
#calculate AUC of model
auc = metrics.roc_auc_score(y_test, xgb_y_pred_proba)
#print AUC score
print(auc)
```

```
0.7139276283237991
```

Plotting feature importance for XGB Classifier

In [59]:
```
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)
```

```
bx.tick_params(axis = 'x', rotation = 90)
```



Model Interpretability - SHAP values

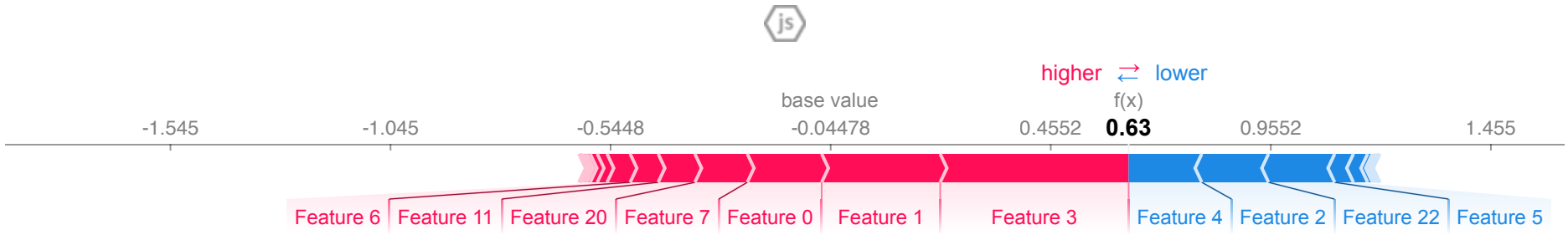
In [60]:
```
import fasttreeshap
shap_explainer = fasttreeshap.TreeExplainer(pipe_xgb, algorithm = "auto", n_jobs = 4)
shap_values = shap_explainer(x_train_os).values
```

In [61]:
```
shap_values
```

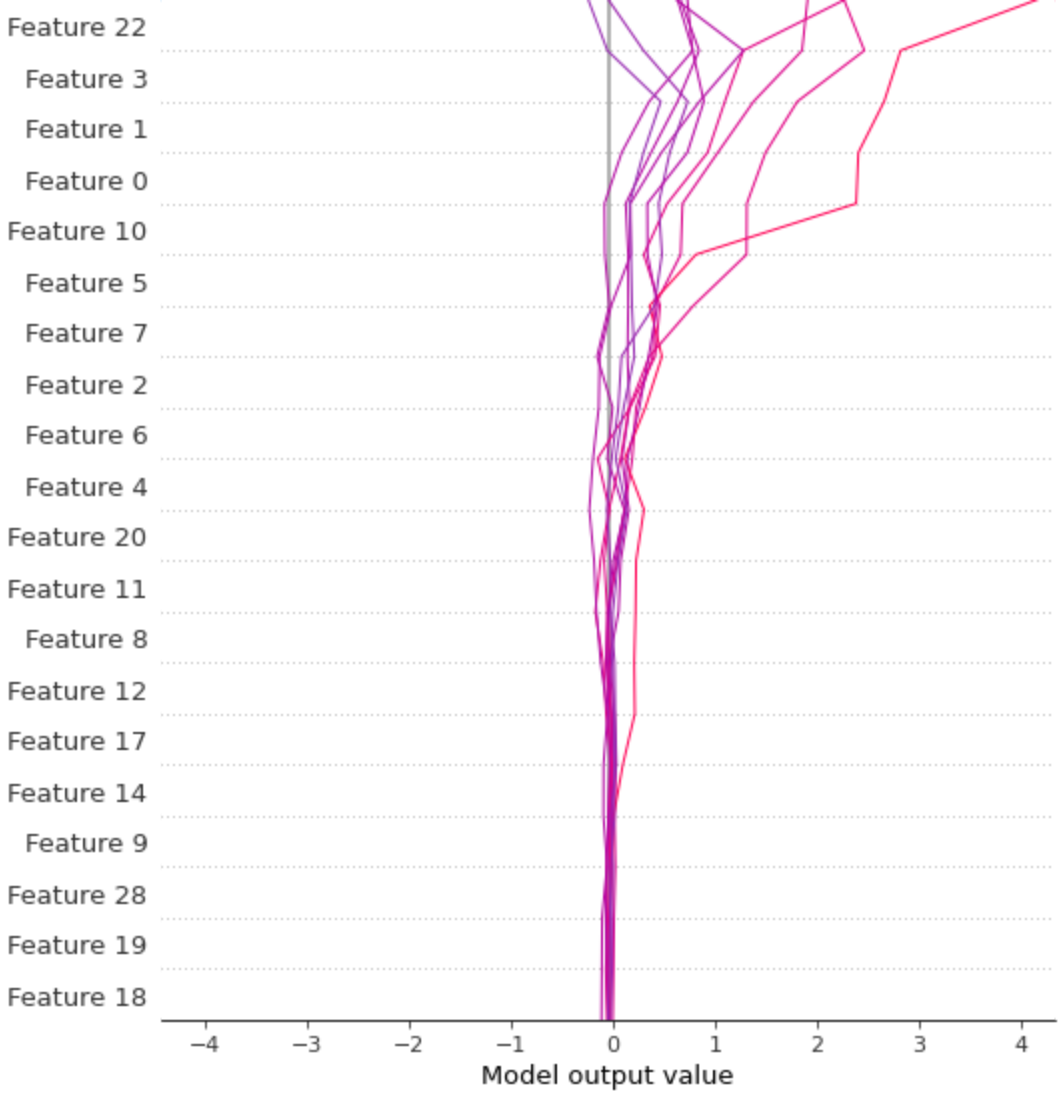
Out[61]:
```
array([[ 1.68950740e-01,  2.69296017e-01, -1.51674977e-01, ...,
        -1.21337111e-03, -2.68879219e-05,  5.96524365e-04],
       [ 3.46242241e-01,  3.50215062e-01,  1.63308681e-01, ...,
        -6.25234310e-04,  2.99746913e-04,  1.45241720e-04],
       [ 2.11504348e-02,  2.52835670e-01,  1.62345359e-01, ...,
        -1.45282440e-03,  1.40389012e-03,  8.17010695e-04],
       ...,
       [ 2.85136001e-01, -2.16957716e+00, -3.91287688e-01, ...,
        -3.62584667e-05,  3.02379437e-04, -3.83267736e-04],
       [-1.52857362e+00, -2.69358478e+00,  1.38289875e-01, ...,
        -3.22744507e-04, -1.08778306e-04, -4.23515639e-04],
       [-1.71449760e+00,  1.26885327e-01, -4.55641811e-01, ...,
         4.20413092e-04, -8.06489862e-04, -3.71293464e-05]])
```

In [70]:
```
shap.initjs()
fasttreeshap.plots.force(shap_explainer.expected_value[0], shap_values[0])
```

Out[70]:



In [75]:
```
fasttreeshap.decision_plot(shap_explainer.expected_value, shap_values[0:10])
```


Feature 22
Feature 3
Feature 1
Feature 0
Feature 10
Feature 5
Feature 7
Feature 2
Feature 6
Feature 4
Feature 20
Feature 11
Feature 8
Feature 12
Feature 17
Feature 14
Feature 9
Feature 28
Feature 19
Feature 18
−4
−3
−2
−1
0
1
2
3
4
Model output value