In [67]:

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import learning_curve
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.svm import SVR
from sklearn.linear_model import Ridge


warnings.filterwarnings('ignore')
```

In [68]:

```
df = pd.read_csv("Data for Cleaning & Modeling.csv")
Meta = pd.read_csv("Metadata.csv")["Definition"]
# df.columns = Meta
df
```

Out[68]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X
0	11.89%	54734.0	80364.0	$25,000	$25,000	$19,080	36 months	B	B4	N
1	10.71%	55742.0	114426.0	$7,000	$7,000	$673	36 months	B	B5	C
2	16.99%	57167.0	137225.0	$25,000	$25,000	$24,725	36 months	D	D3	W Programr
3	13.11%	57245.0	138150.0	$1,200	$1,200	$1,200	36 months	C	C2	city beaum te:
4	13.57%	57416.0	139635.0	$10,800	$10,800	$10,692	36 months	C	C3	State Fa Insurai
...	...	...	...	...	...	...	...	...	...	
399995	12.99%	28753086.0	31226222.0	$10,000	$10,000	$10,000	60 months	C	C1	Administra Assist
399996	16.29%	28753097.0	31226234.0	$13,150	$13,150	$13,150	36 months	D	D2	hel
399997	10.99%	28753099.0	31226236.0	$20,000	$20,000	$20,000	60 months	B	B3	Fac Administra
399998	17.57%	28753118.0	31226256.0	$18,475	$18,475	$18,475	60 months	D	D4	Ser Crea Designer, Sa
399999	13.35%	28753146.0	31226285.0	$16,000	$16,000	$16,000	36 months	C	C2	Electric

400000 rows × 32 columns

In [69]:

```
columns_dict = {"X1": "Interest Rate",
                "X2": "loan_id",
                "X3": "borrower_id",
                "X4": "Loan amount requested",
                "X5": "Loan amount funded",
                "X6": "Investor-funded portion",
                "X7": "Num of payments",
                "X8": "Loan grade",
                "X9": "Loan subgrade",
               "X10": "Employer/job title",
               "X11": "Num of years employed",
               "X12": "Home ownership status",
               "X13": "Annual income of borrower",
               "X14": "Income verified",
               "X15": "Date_issued",
               "X16": "Reason for loan",
               "X17": "Loan category",
               "X18": "Loan title",
               "X19": "Zip code first 3 num",
               "X20": "State of borrower",
               "X21": "Total monthly debt payments / total debt obligations",
               "X22": "Num of past due incidents",
               "X23": "Credit line open date",
               "X24": "Inquiries by creditors during the past 6 months",
               "X25": "Num of months since last delinquency",
               "X26": "Num of months since last public record",
               "X27": "Num of open credit lines",
               "X28": "Num of derogatory public records",
               "X29": "Total credit revolving balance",
               "X30": "Utilization rate",
               "X31": "Total num of current credit lines",
               "X32": "Initial listing status"}
```

In [70]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Data columns (total 32 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   X1      338990 non-null  object
 1   X2      399999 non-null  float64
 2   X3      399999 non-null  float64
 3   X4      399999 non-null  object
 4   X5      399999 non-null  object
 5   X6      399999 non-null  object
 6   X7      399999 non-null  object
 7   X8      338730 non-null  object
 8   X9      338730 non-null  object
 9   X10     376014 non-null  object
 10  X11     382462 non-null  object
 11  X12     338639 non-null  object
 12  X13     338972 non-null  float64
 13  X14     399999 non-null  object
```

In [71]:

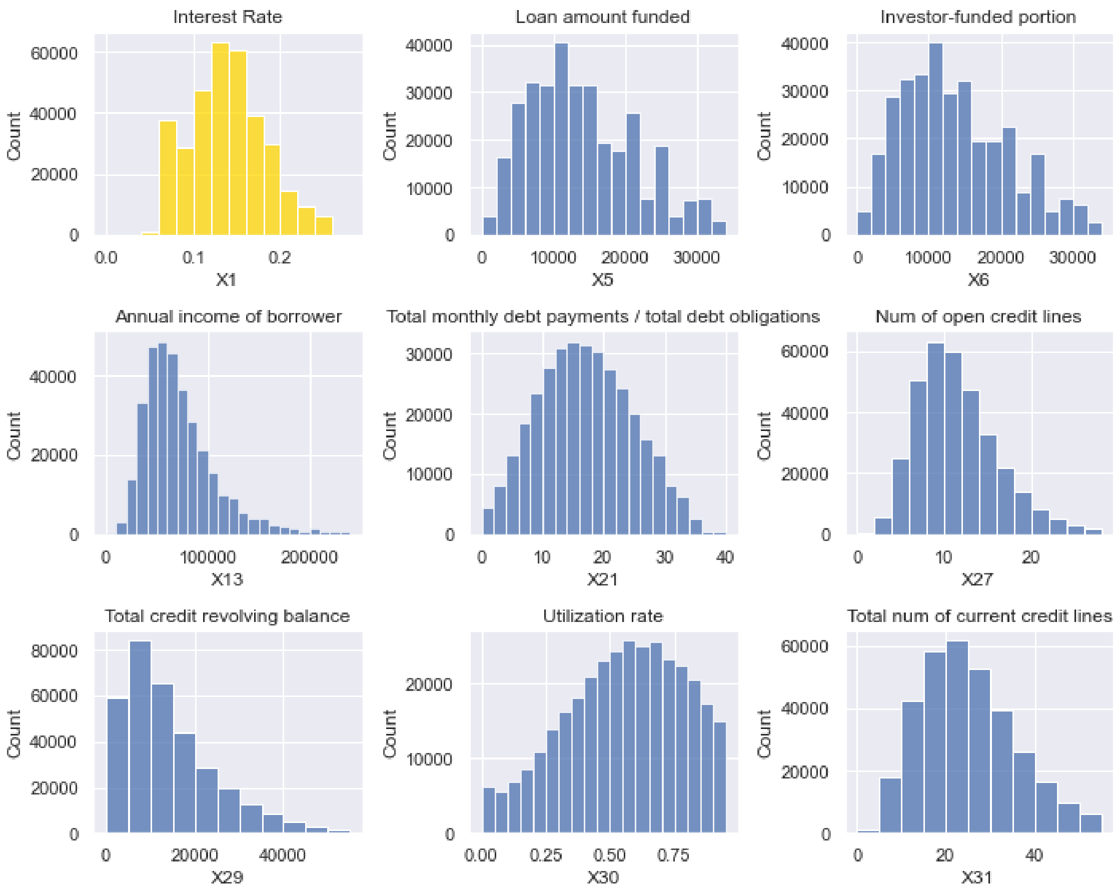
```
# convert to correct data type
df['X1'] = df['X1'].str.replace("%","").astype(float)/100
df['X4'] = df['X4'].str.replace("[$,]","").astype(float)
df['X5'] = df['X5'].str.replace("[$,]","").astype(float)
df['X6'] = df['X6'].str.replace("[$,]","").astype(float)
df['X30'] = df['X30'].str.replace("%","").astype(float)/100
```

In [72]:

```
# interest rate is our target variable, if null, then can't use it for training
df = df.dropna(axis=0, how='any', subset=["X1"])
df["X8"] = df["X8"].fillna(0)
df["X9"] = df["X9"].fillna(0)
df["X10"] = df["X10"].fillna("Unknown")
df["X11"] = df["X11"].fillna("Unknown")
df["X12"] = df["X12"].fillna("Unknown")
df["X13"] = df["X13"].interpolate()
df["X16"] = df["X16"].fillna("Unknown")
df["X18"] = df["X18"].fillna("Unknown")
df["X25"] = df["X25"].fillna(0)
df["X26"] = df["X26"].fillna(0)
df["X30"] = df["X30"].interpolate()
df = df.drop(index=364111)
df = df.reset_index(drop=True)

# df.columns = [columns_dict[i] for i in df.columns]
```

In [73]:

```
fig = plt.figure(figsize=(10,8))
axes = fig.subplots(nrows=4, ncols=2)

plt.subplot(3,3,1)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X1"], bins=np.arange(0,0.3,0.02),color="gold");
plt.title("Interest Rate");

plt.subplot(3,3,2)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X5"], bins=np.arange(0,35000,2000));
plt.title("Loan amount funded");

plt.subplot(3,3,3)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X6"], bins=np.arange(0,35000,2000));
plt.title("Investor-funded portion");

plt.subplot(3,3,4)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X13"], bins=np.arange(0,250000,10000));
plt.title("Annual income of borrower");

plt.subplot(3,3,5)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X21"], bins=np.arange(0,41,2));
plt.title("Total monthly debt payments / total debt obligations");

plt.subplot(3,3,6)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X27"], bins=np.arange(0,30,2));
plt.title("Num of open credit lines");

plt.subplot(3,3,7)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X29"], bins=np.arange(0,60000,5000));
plt.title("Total credit revolving balance");

plt.subplot(3,3,8)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X30"], bins=np.arange(0,1,0.05));
plt.title("Utilization rate");

plt.subplot(3,3,9)
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(df["X31"], bins=np.arange(0,60,5));
plt.title("Total num of current credit lines");

fig.tight_layout()
```



X2 X3: Unique IDs

In [74]:

```
# Have checked there are no duplicates
```

X4 X5: Loan amount requested and funded

In [75]:

```
# create a new label for training: if requested amount is fully funded
# also use one of the loan amount for training
df["is_fully_funded"] = np.where(df["X4"]==df["X5"],1,0)
```

X6

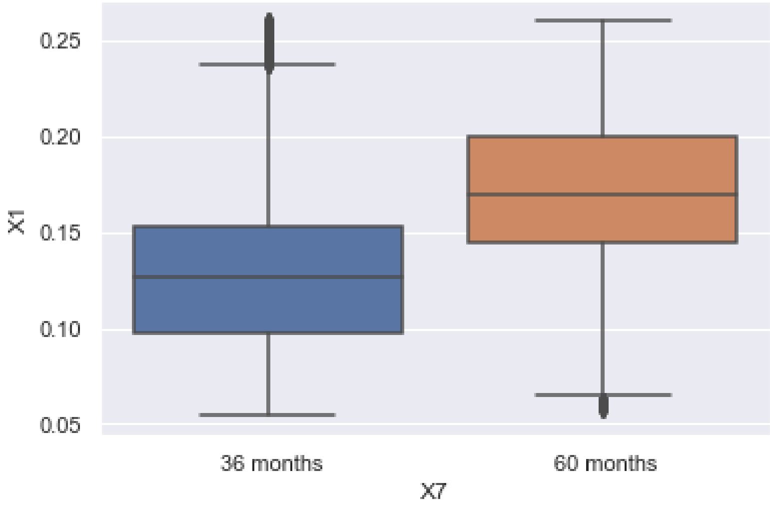
In [76]:

```
# make it a Investor-funded portion of total amount
df["X6"] = df["X6"] / df["X5"]
```

X7

In [77]:

```
# Significant difference, important predictor
sns.set(rc = {'figure.figsize':(6,4)})
sns.boxplot("X7","X1",data=df);
```



X8

In [78]:

```
# I use a X9 which has finer granularity, so drop X8 for not overlapping.
```

X9

In [79]:

```
### siginificant difference
sns.set(rc = {'figure.figsize':(15,4)})
sns.barplot("X9","X1",data=df);
```

X10

In [80]:

```
print("different employment kinds:",len(df["X10"].value_counts()))
df["X10"].value_counts()[0:10]
# too sparse, will easily lead to overfitting
```

```
different employment kinds: 163396
```

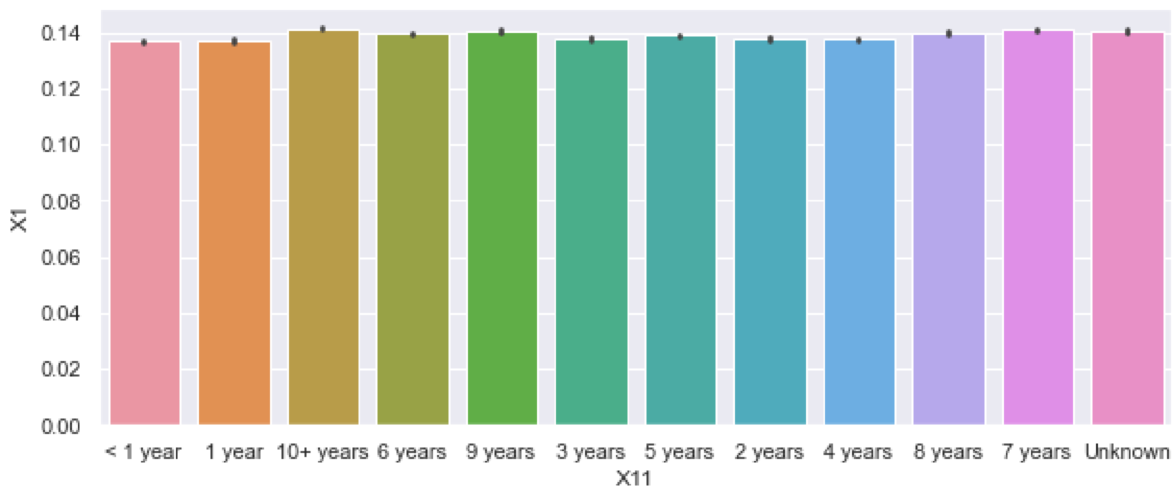
Out[80]:

```
Unknown              20256
Teacher               3602
Manager               2875
Registered Nurse      1537
RN                    1452
Supervisor            1286
Project Manager       1095
Sales                 1048
Office Manager         912
Owner                  870
Name: X10, dtype: int64
```

X11

In [81]:

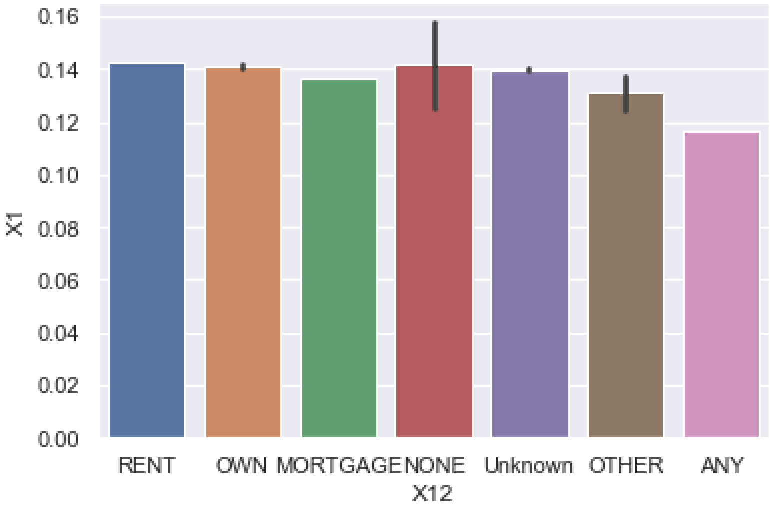
```
# almost no difference, won't have much preditive power.
sns.set(rc = {'figure.figsize':(10,4)})
sns.barplot("X11","X1",data=df);
```



X12

In [82]:

```
# no diff, there's not big difference between mortagage and rent
sns.set(rc = {'figure.figsize':(6,4)})
sns.barplot("X12","X1",data=df);
```



X13 X14

In [83]:

```
# X13 is important predictor by common knowledge, people who have more income have stronger ability
```

In [84]:

```
# X14 is also important, people who can't verify their income will get a lower interest rate.
sns.set(rc = {'figure.figsize':(6,4)})
sns.barplot("X14","X1",data=df);
```

X16 reason for loan

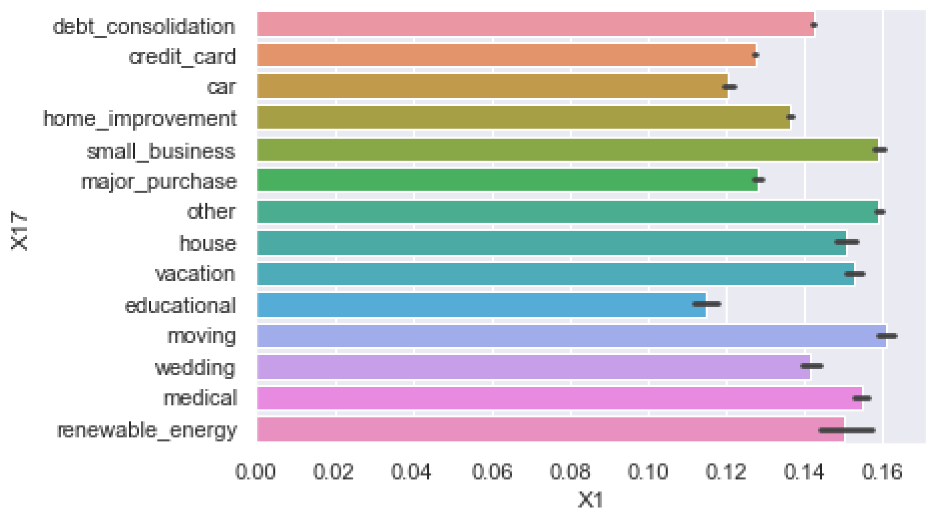
In [85]:

```
# may need some text mining, but I don't think will have too much predicting value
```

X17 loan category

In [86]:

```
# very important predictors as significant difference
sns.set(rc = {'figure.figsize':(6,4)})
sns.barplot(x="X1",y="X17",data=df,orient = 'h');
```



X15 X23 first_loan_since_opened line

In [87]:

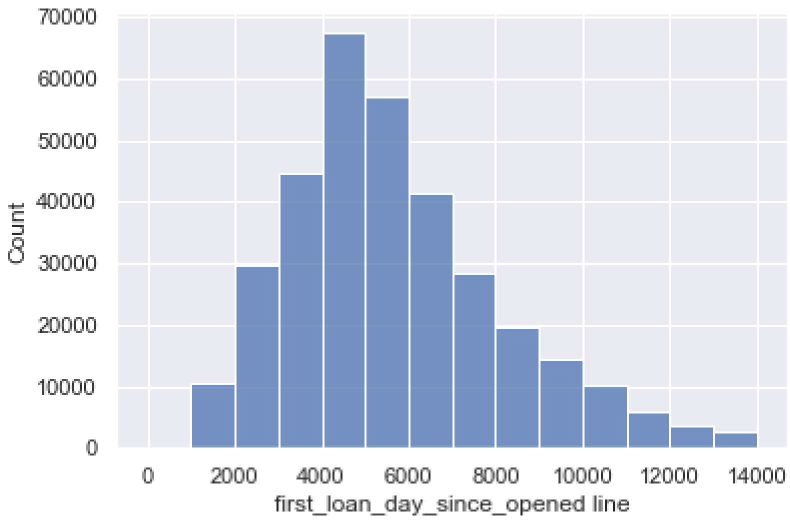
```
# extract first_loan_since_opened line by substract X15 - X23.
from datetime import timedelta
df["X15"] = pd.to_datetime(df["X15"], format="%b-%y").dt.date
df["X23"] = pd.to_datetime(df["X23"], format="%b-%y").dt.date
df["X23"] = df["X23"].apply(lambda x:x-timedelta(days=365*100) if x.year>2043 else x)
```

In [88]:

```
df["first_loan_day_since_opened line"] = df["X15"] -df["X23"]
df["first_loan_day_since_opened line"] = df["first_loan_day_since_opened line"].astype('timedelta64[
```

In [89]:

```
sns.set(rc = {'figure.figsize':(6,4)})
sns.histplot(x="first_loan_day_since_opened line",data=df,bins=np.arange(0,15000,1000));
```



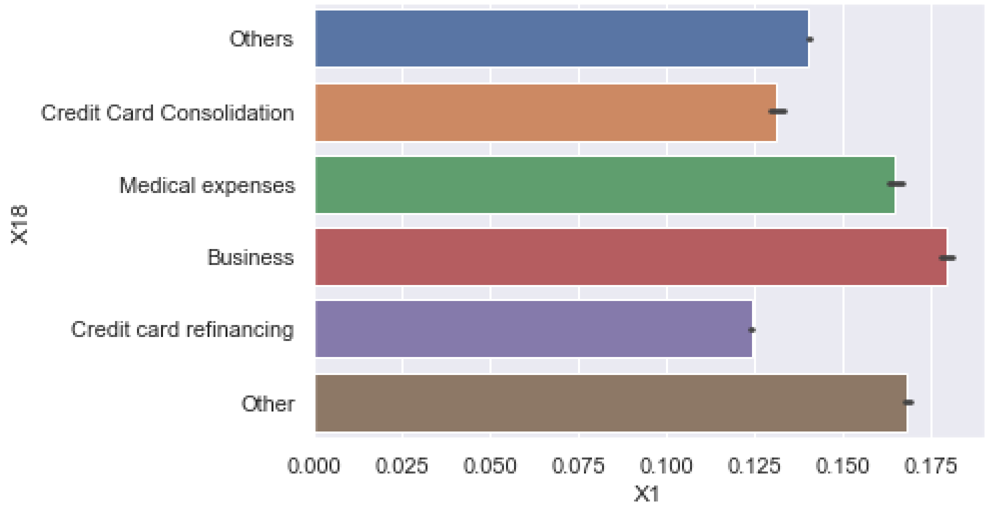
X18 loan title

In [90]:

```
# select frequent-occured loan title with significant difference
def parse_values_X18(x):
    if x not in ["Credit card refinancing","Other","Business","Credit Card Consolidation","Medical
        return "Others"
    else:
        return x

df["X18"] = df['X18'].apply(parse_values_X18)
```

In [91]:

```
sns.set(rc = {'figure.figsize':(6,4)})
sns.barplot(x="X1",y="X18",data=df,orient = 'h');
```



X19 first three number of zip code

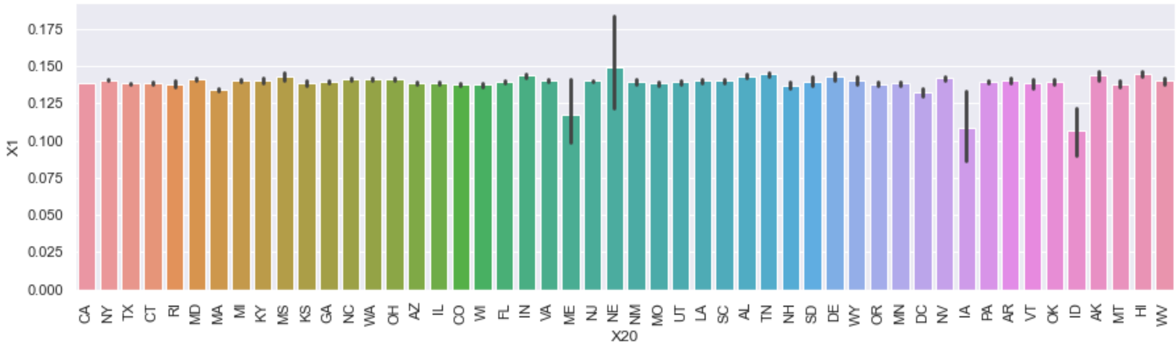
In [92]:

```
# I don't think it has some predictive value with first three digit
# the state may be a better predictor to show geographic value.
# will lead to serious overfitting
```

X20 state

In [93]:

```
sns.set(rc = {'figure.figsize':(15,4)})
sns.barplot("X20","X1",data=df);
plt.xticks(rotation=90);
```



In [94]:

```
df["X20"].value_counts()[["ME","NE","IA","ID"]]

# even though the visualization shows difference,that's out of a small sample.
# so overall no difference for state variable.
```

Out[94]:

```
ME    4
NE    6
IA    7
ID    8
Name: X20, dtype: int64
```

X21 a ratio ...

In [95]:

```
# useful to predict
```

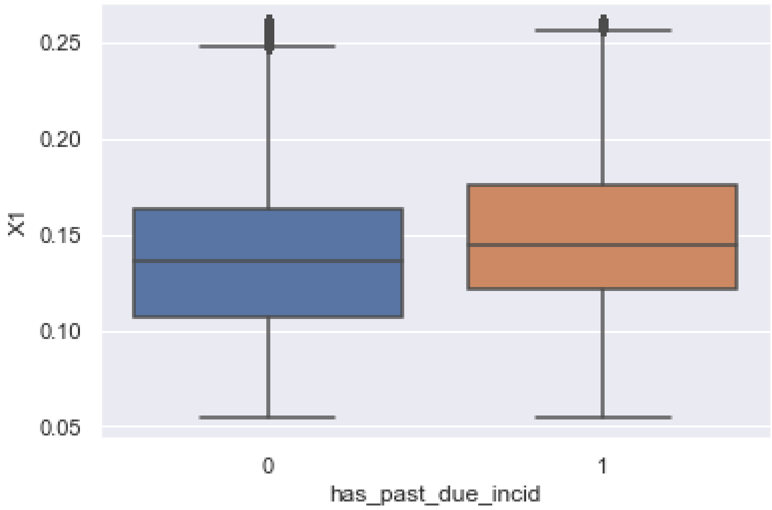
X22 number of 30+ days past-due incidences of delinquency...

In [96]:

```
df["has_past_due_incid"] = np.where(df['X22']>0,1,0)
```

In [97]:

```
sns.set(rc = {'figure.figsize':(6,4)})
sns.boxplot("has_past_due_incid","X1",data=df);
```



X24 X25 X26 some credit reflectors

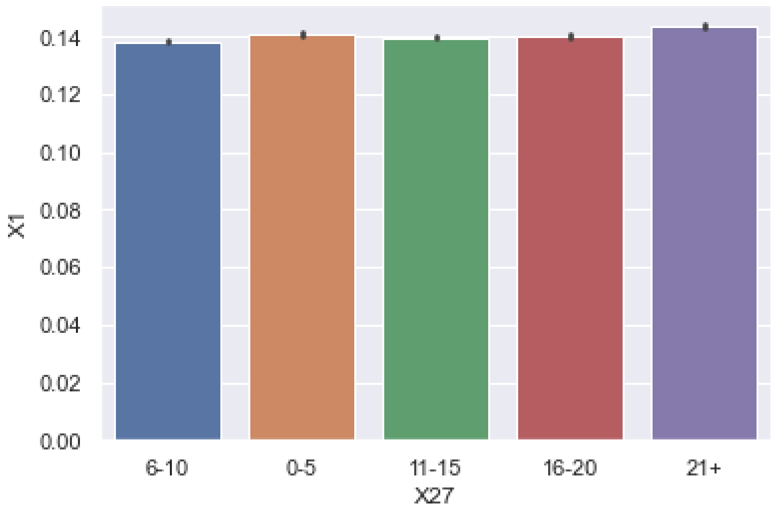
In [98]:

```
### Reflects credit, very important predictors
### replace missing value with 0 since I think it means no incidents occuered
```

X27 number of open credit lines

In [99]:

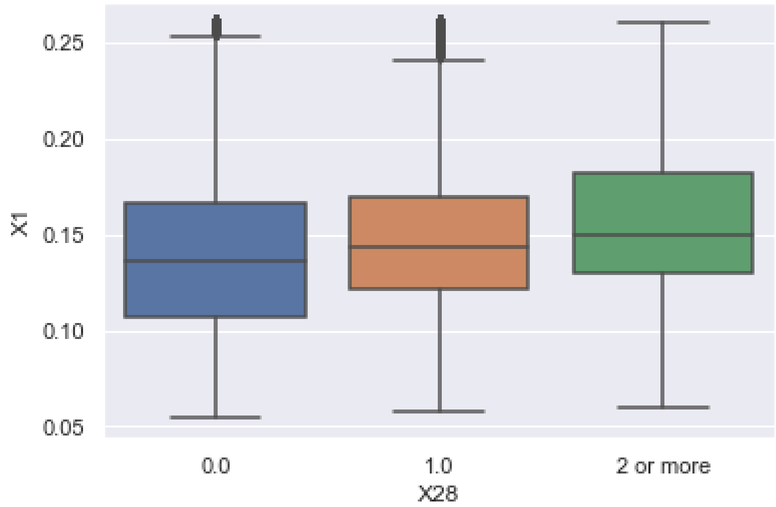
```
sns.set(rc = {'figure.figsize':(6,4)})
def parse_values_X27(x):
    if x <= 5:
        return "0-5"
    elif x>=5 and x<=10:
        return "6-10"
    elif x>10 and x<=15:
        return "11-15"
    elif x>15 and x<=20:
        return "16-20"
    else:
        return "21+"

df["X27"] = df['X27'].apply(parse_values_X27)

sns.barplot("X27","X1",data=df);
```



X28 number of derogatory public records

In [100]:

```
sns.set(rc = {'figure.figsize':(6,4)})
df["X28"] = df["X28"].apply(lambda x:x if x<2 else "2 or more")
sns.boxplot("X28","X1",data=df);
```



X29 X30 revolving balance and utilization rate...

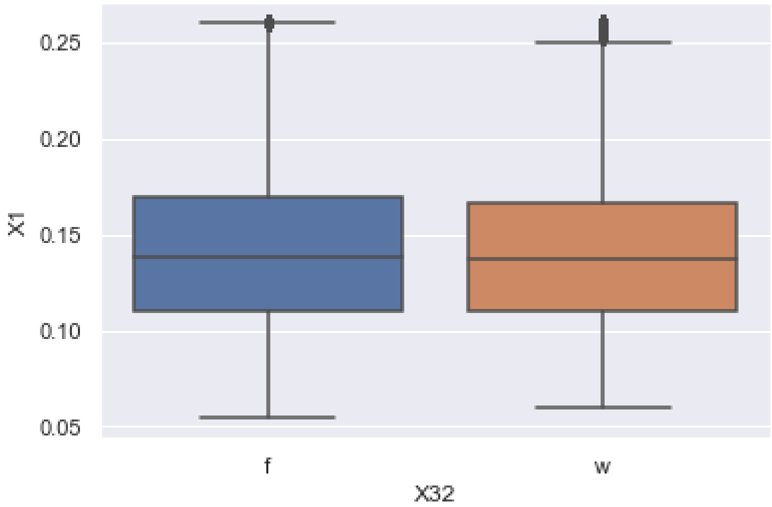
In [101]:

```
### Useful in predicting. More balance means more ability to pay back.
```

32 initial listing status

In [102]:

```
# Acutually almost no difference, won't have too much predictive power.
sns.set(rc = {'figure.figsize':(6,4)})
sns.boxplot("X32","X1",data=df);
```



In [ ]:

```
```

In [106]:

```
use_cols = ["is_fully_funded","has_past_due_incid",
            "X5","X6","X7","X9","X11","X12","X13","X14","X17","X18","X21","X22","X24","X25","X26","
X = df[use_cols]
y = df["X1"]
```

In [107]:

```
X.columns
```

Out[107]:

```
Index(['is_fully_funded', 'has_past_due_incid', 'X5', 'X6', 'X7', 'X9', 'X11',
       'X12', 'X13', 'X14', 'X17', 'X18', 'X21', 'X22', 'X24', 'X25', 'X26',
       'X27', 'X28', 'X29', 'X30', 'X31', 'X32'],
      dtype='object')
```

In [108]:

```
# get dummies
X = pd.get_dummies(X,drop_first=True).drop(["X11_Unknown","X12_Unknown"],axis=1)
```

In [109]:

```
X.columns
```

Out[109]:

```
Index(['is_fully_funded', 'has_past_due_incid', 'X5', 'X6', 'X13', 'X21',
       'X22', 'X24', 'X25', 'X26', 'X29', 'X30', 'X31', 'X7_ 60 months',
       'X9_A1', 'X9_A2', 'X9_A3', 'X9_A4', 'X9_A5', 'X9_B1', 'X9_B2', 'X9_B3',
       'X9_B4', 'X9_B5', 'X9_C1', 'X9_C2', 'X9_C3', 'X9_C4', 'X9_C5', 'X9_D1',
       'X9_D2', 'X9_D3', 'X9_D4', 'X9_D5', 'X9_E1', 'X9_E2', 'X9_E3', 'X9_E4',
       'X9_E5', 'X9_F1', 'X9_F2', 'X9_F3', 'X9_F4', 'X9_F5', 'X9_G1', 'X9_G2',
       'X9_G3', 'X9_G4', 'X9_G5', 'X11_10+ years', 'X11_2 years',
       'X11_3 years', 'X11_4 years', 'X11_5 years', 'X11_6 years',
       'X11_7 years', 'X11_8 years', 'X11_9 years', 'X11_< 1 year',
       'X12_MORTGAGE', 'X12_NONE', 'X12_OTHER', 'X12_OWN', 'X12_RENT',
       'X14_VERIFIED - income source', 'X14_not verified', 'X17_credit_card',
       'X17_debt_consolidation', 'X17_educational', 'X17_home_improvement',
       'X17_house', 'X17_major_purchase', 'X17_medical', 'X17_moving',
       'X17_other', 'X17_renewable_energy', 'X17_small_business',
       'X17_vacation', 'X17_wedding', 'X18_Credit Card Consolidation',
       'X18_Credit card refinancing', 'X18_Medical expenses', 'X18_Other',
       'X18_Others', 'X27_11-15', 'X27_16-20', 'X27_21+', 'X27_6-10',
       'X28_1.0', 'X28_2 or more', 'X32_w'],
      dtype='object')
```

In [110]:

```
len(X.columns)
```

Out[110]:

```
91
```

train and split

In [111]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8,test_size=0.2)
```

StandardScaler

In [112]:

```
scalerX = StandardScaler()
scalerX.fit(X_train)

X_train = scalerX.transform(X_train)
X_test = scalerX.transform(X_test)
```

cross validation & learning curve

In [113]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve_r2(estimator, title, X, y, ylim=None, cv=None, n_jobs=1,
                        train_sizes=np.linspace(.05, 1., 20), verbose=0, plot=True):

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, verbose=verbose)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    if plot:
        plt.figure()
        plt.title(title)
        if ylim is not None:
            plt.ylim(*ylim)
        plt.xlabel("traningsize")
        plt.ylabel("score")
        #plt.gca().invert_yaxis()
        plt.grid()

        plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + trai
                         alpha=0.1, color="b")
        plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_sc
                         alpha=0.1, color="r")
        plt.plot(train_sizes, train_scores_mean, 'o-', color="b", label="train score")
        plt.plot(train_sizes, test_scores_mean, 'o-', color="r", label="test score")

        plt.legend(loc="best")

        plt.draw()
        plt.show()
        #plt.gca().invert_yaxis()

    midpoint = ((train_scores_mean[-1] + train_scores_std[-1]) + (test_scores_mean[-1] - test_score
    diff = (train_scores_mean[-1] + train_scores_std[-1]) - (test_scores_mean[-1] - test_scores_std
    return midpoint, diff
```

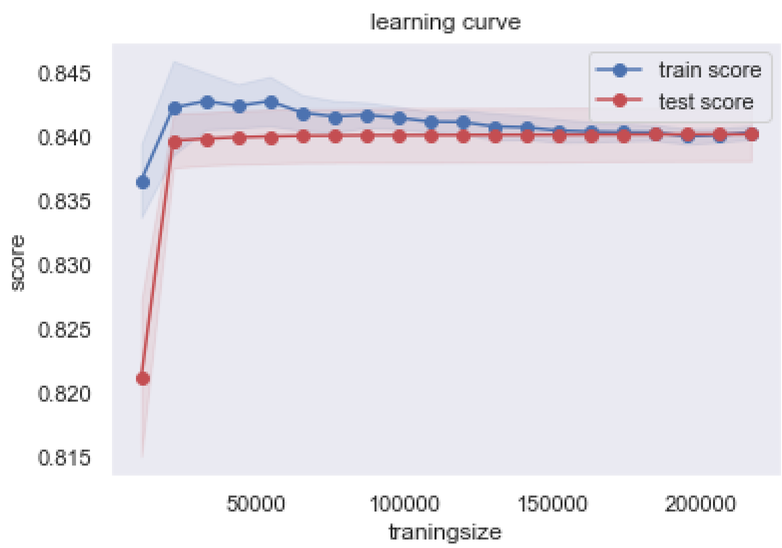
In [114]:

```
def cv_learning_curve_loss(clf,k):
    cv = KFold(n_splits=k)
    cv_scores = cross_val_score(clf,X_train,y_train,cv=cv) # Storing the CV scores of each fold
    print("cv_scores",np.mean(cv_scores))

    # build learning curve
    train_size,train_scores,test_scores = learning_curve(clf,X_train,y=y_train,scoring="neg_mean_squ
    train_scores = np.mean(train_scores,axis=1)
    test_scores = np.mean(test_scores,axis=1)
    lc = pd.DataFrame({"Training_size":train_size,"Training_loss":train_scores,"Validation_loss":tes

    print("Mean train loss",np.mean(train_scores))
    print("Mean test loss",np.mean(test_scores))


    # plot learning curve
    sns.lineplot(data=lc,x="Training_size",y="value",hue="variable")
    plt.title("Learning Curve")
    plt.ylabel("neg_mse");
```

Linear Regression

In [56]:

```
plot_learning_curve_r2(LinearRegression(), "learning curve", X_train, y_train)
```



Out[56]:

(0.8394458799803648, 0.002735025785587042)

In [123]:

```
cv_learning_curve_loss(LinearRegression(),10)
```

```
cv_scores 0.8408503858726937
Mean train loss -0.0003025450765515491
Mean test loss -0.0003097356963580475
```

SVM with GridSearch

In [145]:

```
parameters = {'kernel': ('linear', 'rbf','poly'), 'C':[0.01,0.1,1],'gamma': [1e-7, 1e-4],'epsilon':[
svr = svm.SVR()
grid_svr = GridSearchCV(svr, parameters)
grid_svr.fit(X_train,y_train)
grid_svr.best_params_
```

Out[145]:

```
{'C': 0.01, 'epsilon': 0.1, 'gamma': 1e-07, 'kernel': 'linear'}
```

In [146]:

```
best_model_svr = grid_svr.best_estimator_
best_model_svr
```

Out[146]:

```
SVR(C=0.01, gamma=1e-07, kernel='linear')
```

In [147]:

```
best_model_svr = SVR(kernel="linear",C = 0.1,gamma = 1e-07)
best_model_svr.fit(X_train, y_train)
y_test_pred_svm = best_model_svr.predict(X_test)
print("RMSE",mean_squared_error(y_test,y_test_pred_svm,squared=False))
```

RMSE 0.04540621971574321

Ridge Regression with GridSearch

In [615]:

```
parameters = {'alpha':[1,10,20,30,40,50]}
model = Ridge()
Ridge_reg= GridSearchCV(model, parameters, scoring='neg_mean_squared_error',cv=10)
Ridge_reg.fit(X_train,y_train)
print(Ridge_reg.best_estimator_)
print(Ridge_reg.best_estimator_.alpha)
print(Ridge_reg.best_score_)

# best model
best_model = Ridge_reg.best_estimator_
best_model.fit(X_train,y_train)
```

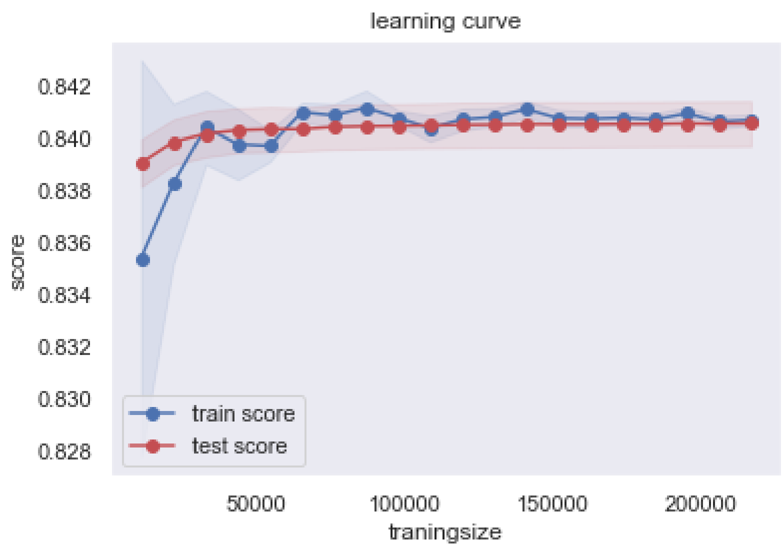
Ridge(alpha=20)
20
-0.0003072293445232321

Out[615]:

Ridge(alpha=20)

In [714]:

```
plot_learning_curve_r2(Ridge(alpha=20), "learning curve", X_train, y_train)
```



Out[714]:

(0.8402617821986468, 0.0012084864110897264)

pros and cons of model I used

I used ridge regession at last which performs 0.84 in R2. Pros is that the model converges well.The ridge regression uses the parameter alpha to regularize coefficient estimates to prevent overfitting, which is a big advantage over linear regression. Also, the model is not complex:I use GridSearch to select the best parameter,and has a good explainality. Another advantage of ridge regression is not requiring unbiased estimators. For cons, it trades variance for bias, and it includes all predictors in the final model. Also, I think more advanced can be attempted further.

testset Peformance (final model: ridge regression)

In [746]:

```
y_test_pred_ridge = best_model.predict(X_test)
y_test_pred_ridge

print("r2",r2_score(y_test,  y_test_pred_ridge))
print("RMSE",mean_squared_error(y_test,y_test_pred_ridge,squared=False))
```

```
r2 0.838342661894117
RMSE 0.017571404835510014
```

holdout Test Output

In [855]:

```
test = pd.read_csv("Holdout for Testing.csv")
```

In [856]:

```
test['X4'] = test['X4'].str.replace("[$,]","").astype(float)
test['X5'] = test['X5'].str.replace("[$,]","").astype(float)
test['X6'] = test['X6'].str.replace("[$,]","").astype(float)
test['X30'] = test['X30'].str.replace("%","").astype(float)/100
test["X8"] = test["X8"].fillna(0)
test["X9"] = test["X9"].fillna(0)
test["X10"] = test["X10"].fillna("Unknown")
test["X11"] = test["X11"].fillna("Unknown")
test["X12"] = test["X12"].fillna("Unknown")
test["X13"] = test["X13"].interpolate()
test["X16"] = test["X16"].fillna("Unknown")
test["X18"] = test["X18"].fillna("Unknown")
test["X25"] = test["X25"].fillna(0)
test["X26"] = test["X26"].fillna(0)
test["X30"] = test["X30"].interpolate()
test = test.reset_index(drop=True)
```

In [857]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80000 entries, 0 to 79999
Data columns (total 32 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   X1      0 non-null      float64
 1   X2      80000 non-null  int64  
 2   X3      80000 non-null  int64  
 3   X4      80000 non-null  float64
 4   X5      80000 non-null  float64
 5   X6      80000 non-null  float64
 6   X7      80000 non-null  object 
 7   X8      80000 non-null  object 
 8   X9      80000 non-null  object 
 9   X10     80000 non-null  object 
 10  X11     80000 non-null  object 
 11  X12     80000 non-null  object 
 12  X13     80000 non-null  float64
 13  X14     80000 non-null  object 
 14  X15     80000 non-null  object 
 15  X16     80000 non-null  object 
 16  X17     80000 non-null  object 
 17  X18     80000 non-null  object 
 18  X19     80000 non-null  object 
 19  X20     80000 non-null  object 
 20  X21     80000 non-null  float64
 21  X22     80000 non-null  int64  
 22  X23     80000 non-null  object 
 23  X24     80000 non-null  int64  
 24  X25     80000 non-null  float64
 25  X26     80000 non-null  float64
 26  X27     80000 non-null  int64  
 27  X28     80000 non-null  int64  
 28  X29     80000 non-null  int64  
 29  X30     80000 non-null  float64
 30  X31     80000 non-null  int64  
 31  X32     80000 non-null  object 
dtypes: float64(9), int64(8), object(15)
memory usage: 19.5+ MB
```

In [858]:

```
test["is_fully_funded"] = np.where(test["X4"]==test["X5"],1,0)
test["X6"] = test["X6"] / test["X5"]
# test["X15"] = pd.to_datetime(test["X15"], format="%y-%b").dt.date
# test["X23"] = pd.to_datetime(test["X23"], format="%y-%b").dt.date
# test["X23"] = test["X23"].apply(lambda x:x-timedelta(days=365*100) if x.year>2043 else x)
# test["first_loan_day_since_opened line"] = test["X15"] -test["X23"]
# test["first_loan_day_since_opened line"] = test["first_loan_day_since_opened line"].astype('timede

test["X18"] = test['X18'].apply(parse_values_X18)
test["has_past_due_incid"] = np.where(test['X22']>0,1,0)

test["X27"] = test['X27'].apply(parse_values_X27)
test["X28"] = test["X28"].apply(lambda x:x if x<2 else "2 or more")
```

In [860]:

```
use_cols = ["is_fully_funded","has_past_due_incid",
            "X5","X6","X7","X9","X11","X12","X13","X14","X17","X18","X21","X22","X24","X25","X26","
test_X = test[use_cols]
test_y = test["X1"]

test_X = pd.get_dummies(test_X,drop_first=True).drop(["X11_Unknown"],axis=1)

test_X.rename(columns={"X28_1":"X28_1.0"},inplace=True)
test_X.columns
# join a missing column
test_X.insert(0, 'X18_Credit Card Consolidation', 0)
test_X.insert(0, 'X12_OTHER', 0)
test_X.insert(0, 'X17_educational', 0)
test_X.insert(0, 'X12_MORTGAGE', 0)
test_X.insert(0, 'X12_NONE', 0)
test_X.insert(0, 'X9_A1', 0)
```

In [863]:

```
scalerX = StandardScaler()
scalerX.fit(test_X)
test_X = scalerX.transform(test_X)
test_X
y_test_pred = best_model.predict(test_X)
y_test_pred
```

Out[863]:

```
array([0.15600854, 0.13999419, 0.15947333, ..., 0.16427441, 0.15203303,
       0.13167085])
```

In [865]:

```
test["X1"] = y_test_pred
```

In [869]:

```
test.to_csv("test_loan.csv",index=False)
```