lab 9 - Jackson Nahom

October 24, 2021

[1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn import metrics
```

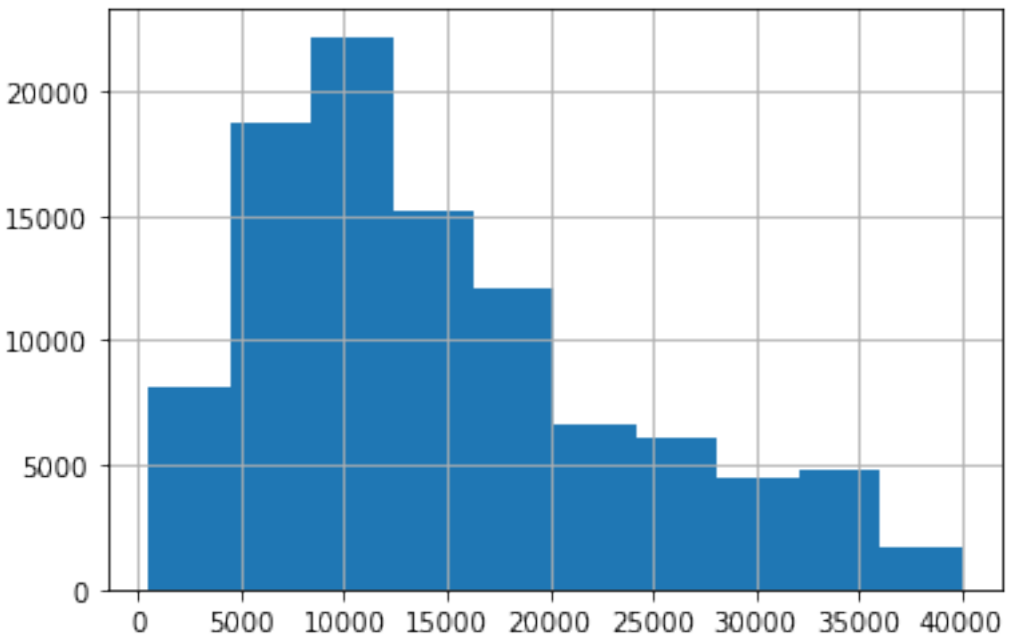
[2]:
```
ld = pd.read_csv('data/lendingclub_2015-2018.csv')
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns
(20,60,119,130,131,132,135,136,137,140,146,147,148) have mixed types.Specify
dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[3]:
```
ld = ld.sample(100000, random_state=516)
```

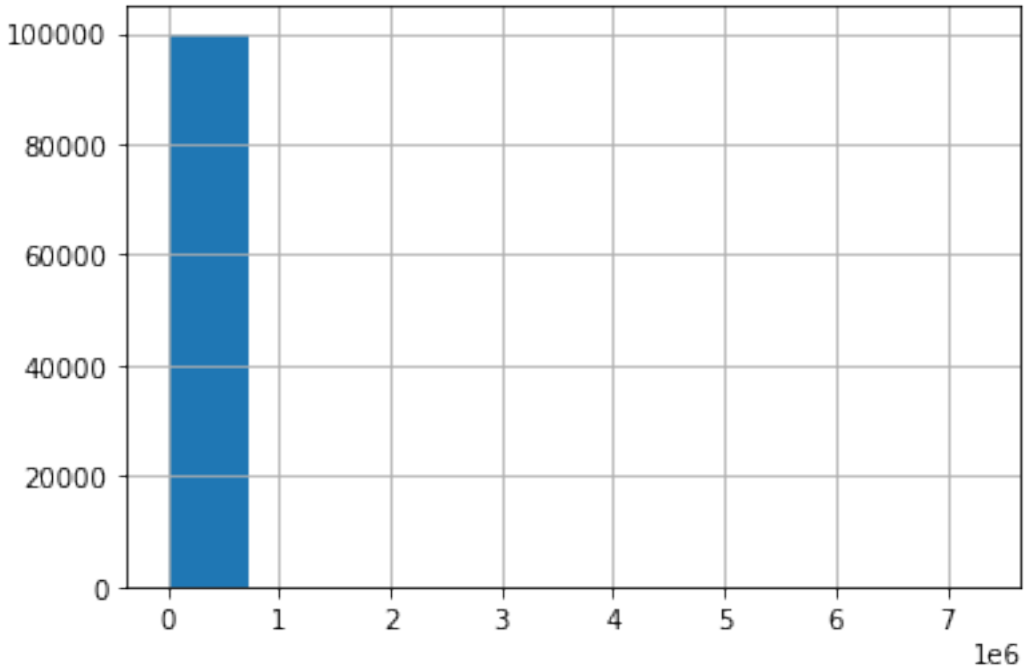
[4]:
```
ld['loan_amnt'].hist()
```

[4]: <AxesSubplot:>



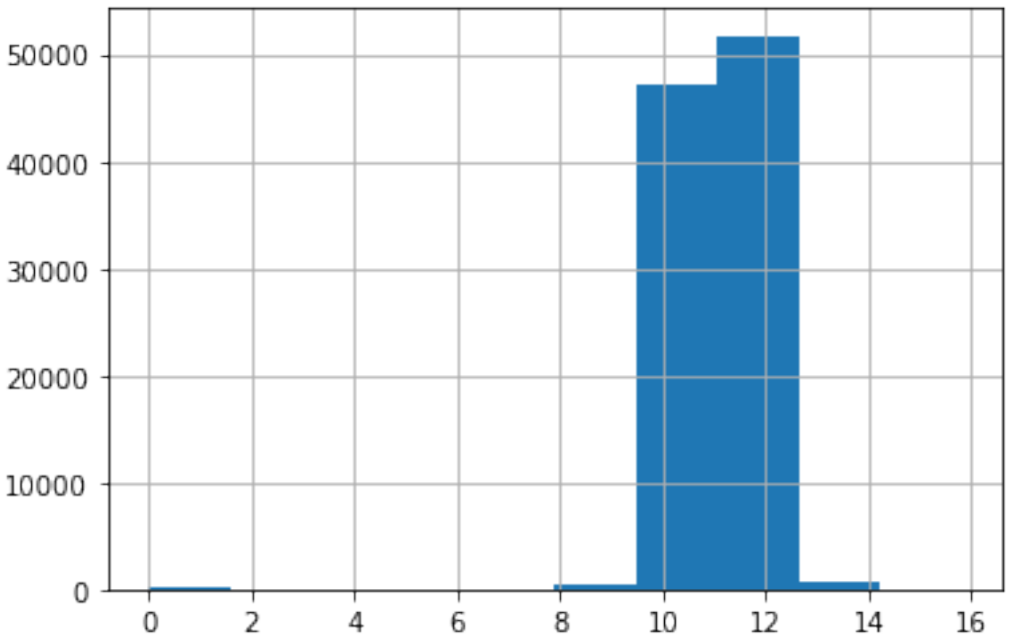
[5]:
```
ld['annual_inc'].hist()
```

[5]: <AxesSubplot:>



[6]:
```
ld['log_annual_inc'] = np.log(ld['annual_inc']+1)
ld['log_annual_inc'].hist()
```

[6]: <AxesSubplot:>



[7]:
```
# view unique values
ld['term'].unique()

# split rows into parts
term_split = ld['term'].str.split(' ')

# view first five rows
print(term_split[:5])
```

```
138233    [, 36, months]
75960     [, 36, months]
58028     [, 60, months]
42825     [, 60, months]
352       [, 36, months]
Name: term, dtype: object
```

[8]:
```
# the str function can retrieve a specific list element for all rows
term_split.str[1]
ld['duration'] = term_split.str[1]

# add this to the dataframe
display(ld['duration'].head())
# this column is not in integer format. Must fix!
```

```
138233    36
75960     36
58028     60
```

```
42825     60
352       36
Name: duration, dtype: object
```

[9]:
```
# convert column to integer
ld['duration'] = ld['duration'].apply(int)
display(ld['duration'].head())
```

```
138233    36
75960     36
58028     60
42825     60
352       36
Name: duration, dtype: int64
```

[10]:
```
cols = ['int_rate', 'loan_amnt', 'installment', 'log_annual_inc', 'duration',␣
 ↪'fico_range_low', 'revol_util', 'dti']
corr = ld[cols].corr()
corr.style.background_gradient(cmap='coolwarm')

# ld[cols].corr() # <--- use this if you just want the table in non-graphical␣
 ↪format
```

[10]:
```
<pandas.io.formats.style.Styler at 0x1cd00e75370>
```

[11]:
```
pred_vars = ['loan_amnt', 'log_annual_inc', 'fico_range_low', 'revol_util',␣
 ↪'dti', 'duration']
```

[12]:
```
print("before dropping rows with missing data", len(ld))
ld = ld.dropna(subset=pred_vars)
print("after dropping rows with missing data", len(ld))
```

```
before dropping rows with missing data 100000
after dropping rows with missing data 99822
```

[13]:
```
from sklearn.model_selection import train_test_split

# use index-based sampling since we have time series data
train, test = train_test_split(ld, test_size=0.25, shuffle=False)
```

[14]:
```
# earliest and latest dates in train
print("training data starts\n", train['issue_d'].head())
print("training data ends\n", train['issue_d'].tail())
# earliest and latest in test
print("testing data starts\n", test['issue_d'].head())
print("testing data ends\n", test['issue_d'].tail())
```

```
training data starts
 138233    Oct-2016
```

```
75960     Aug-2015
58028     Feb-2015
42825     Sep-2014
352       Jan-2009
Name: issue_d, dtype: object
training data ends
 126476    Jul-2016
37791     Jun-2014
125601    Jul-2016
88562     Oct-2015
68643     May-2015
Name: issue_d, dtype: object
testing data starts
 230       Jun-2008
110235    Apr-2016
239195    Oct-2018
232688    Sep-2018
237085    Oct-2018
Name: issue_d, dtype: object
testing data ends
 113860    Apr-2016
91441     Dec-2015
91903     Nov-2015
119500    May-2016
229221    Aug-2018
Name: issue_d, dtype: object
```

[15]:
```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor()

rf_reg.fit(train[pred_vars], train['int_rate'])
```

[15]:
```
RandomForestRegressor()
```

[19]:
```
reg_multi = sm.OLS(train['int_rate'], train[pred_vars], hasconst=False).fit()
reg_multi.summary()
```

[19]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                int_rate   R-squared (uncentered):
0.904
Model:                            OLS   Adj. R-squared (uncentered):
0.904
Method:                 Least Squares   F-statistic:
```

```
1.182e+05
Date:                Wed, 20 Oct 2021   Prob (F-statistic):
0.00
Time:                        18:25:27   Log-Likelihood:
-2.1554e+05
No. Observations:               74866   AIC:
4.311e+05
Df Residuals:                   74860   BIC:
4.311e+05
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
loan_amnt      -6.867e-05   1.95e-06    -35.162      0.000   -7.25e-05
-6.48e-05
log_annual_inc     0.9998      0.026     38.879      0.000       0.949
1.050
fico_range_low    -0.0115      0.000    -29.707      0.000      -0.012
-0.011
revol_util         0.0416      0.001     59.439      0.000       0.040
0.043
dti                0.0364      0.001     33.717      0.000       0.034
0.038
duration           0.1915      0.002    122.819      0.000       0.188
0.195
==============================================================================
Omnibus:                     8092.942   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            12609.976
Skew:                           0.793   Prob(JB):                         0.00
Kurtosis:                       4.236   Cond. No.                     2.88e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 2.88e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[16]:
```
from sklearn.svm import SVR
```

```
svr_reg = SVR()

svr_reg.fit(train[pred_vars], train['int_rate'])
```

[16]:
```
SVR()
```

[17]:
```
from sklearn.neural_network import MLPRegressor

mlp_reg = MLPRegressor()

mlp_reg.fit(train[pred_vars], train['int_rate'])
```

[17]:
```
MLPRegressor()
```

[20]:
```
models = [reg_multi, rf_reg, svr_reg, mlp_reg]

for reg in models:

    reg_pred = reg.predict(test[pred_vars])

    reg_rmse = metrics.mean_squared_error(test['int_rate'], reg_pred,␣
 ↪squared=False)
    print(reg, "RMSE:", reg_rmse)
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper object at
0x000001CD003162E0> RMSE: 4.312693552331951
RandomForestRegressor() RMSE: 3.9174048997664697
SVR() RMSE: 4.7771657073737925
MLPRegressor() RMSE: 4.964039557015436
```

1 Tasks

1. Use another algorithm for regression (See this list) https://scikit-learn.org/stable/supervised_learning.html. Compare the models again. Which performed best?
2. In last week's lab, you were asked to add additional variables to try improving the predictions. Use those variables again on each of these models and evaluate. How does the RMSE change with the additional predictors? Which model was best?

1.0.1 Task 1

Use another algorithm for regression (See this list) https://scikit-learn.org/stable/supervised_learning.html. Compare the models again. Which performed best?

[49]:
```
from sklearn.cross_decomposition import PLSRegression
```

```
pls2 = PLSRegression(n_components=6, max_iter=500, scale=True, copy=True)
pls2.fit(train[pred_vars], train['int_rate'])
```

[49]:
```
PLSRegression(copy=False, n_components=6)
```

[52]:
```
models = [reg_multi, rf_reg, svr_reg, mlp_reg, pls2]

for reg in models:

    reg_pred = reg.predict(test[pred_vars])

    reg_rmse = metrics.mean_squared_error(test['int_rate'], reg_pred,␣
 ↪squared=False)
    print(reg, "RMSE:", reg_rmse)
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper object at
0x000001CD003162E0> RMSE: 4.312693552331951
RandomForestRegressor() RMSE: 3.9174048997664697
SVR() RMSE: 4.7771657073737925
MLPRegressor() RMSE: 4.964039557015436
PLSRegression(copy=False, n_components=6) RMSE: 3.928986485391665
```

[53]:
```
print('rf_reg difference with pls2 ', 3.928986485391665-3.9174048997664697)
```

```
rf_reg difference with pls2  0.011581585625195512
```

In favor of rf_reg

2 Task 2

In last week's lab, you were asked to add additional variables to try improving the predictions. Use those variables again on each of these models and evaluate. How does the RMSE change with the additional predictors? Which model was best?

[60]:
```
pred_vars_2 = ['log_funded_amnt', 'inq_last_6mths', 'open_acc', 'bc_util',␣
 ↪'log_last_pymnt_amnt'
               ,'duration']
```

[61]:
```
ld = pd.read_csv('data/lendingclub_2015-2018.csv')
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns
(20,60,119,130,131,132,135,136,137,140,146,147,148) have mixed types.Specify
dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[62]:
```
ld = ld.sample(100000, random_state=516)
```

[63]:
```
ld['log_funded_amnt'] = np.log(ld['funded_amnt'])
ld['log_last_pymnt_amnt'] = np.log(ld['last_pymnt_amnt']+1)
```

[64]:
```
print("before dropping rows with missing data", len(ld))
ld = ld.dropna(subset=pred_vars_2)
print("after dropping rows with missing data", len(ld))
```

```
before dropping rows with missing data 100000
after dropping rows with missing data 96505
```

[65]:
```
# view unique values
ld['term'].unique()

# split rows into parts
term_split = ld['term'].str.split(' ')

# view first five rows
print(term_split[:5])
```

```
138233    [, 36, months]
75960     [, 36, months]
58028     [, 60, months]
42825     [, 60, months]
108763    [, 36, months]
Name: term, dtype: object
```

[66]:
```
# the str function can retrieve a specific list element for all rows
term_split.str[1]
ld['duration'] = term_split.str[1]

# add this to the dataframe
display(ld['duration'].head())
# this column is not in integer format. Must fix!
```

```
138233    36
75960     36
58028     60
42825     60
108763    36
Name: duration, dtype: object
```

[67]:
```
# convert column to integer
ld['duration'] = ld['duration'].apply(int)
display(ld['duration'].head())
```

```
138233    36
75960     36
58028     60
42825     60
```

```
108763    36
Name: duration, dtype: int64
```

[68]:
```
corr = ld[pred_vars_2].corr()
corr.style.background_gradient(cmap='coolwarm')
```

[68]:
```
<pandas.io.formats.style.Styler at 0x1cd00ee0400>
```

[69]:
```
train, test = train_test_split(ld, test_size=0.25, shuffle=False)
```

[70]:
```
# earliest and latest dates in train
print("training data starts\n", train['issue_d'].head())
print("training data ends\n", train['issue_d'].tail())
# earliest and latest in test
print("testing data starts\n", test['issue_d'].head())
print("testing data ends\n", test['issue_d'].tail())
```

```
training data starts
 138233    Oct-2016
75960     Aug-2015
58028     Feb-2015
42825     Sep-2014
108763    Mar-2016
Name: issue_d, dtype: object
training data ends
 150390    Feb-2017
45238     Oct-2014
240174    Oct-2018
159432    May-2017
245692    Nov-2018
Name: issue_d, dtype: object
testing data starts
 138753    Oct-2016
58207     Mar-2015
134891    Sep-2016
126476    Jul-2016
37791     Jun-2014
Name: issue_d, dtype: object
testing data ends
 113860    Apr-2016
91441     Dec-2015
91903     Nov-2015
119500    May-2016
229221    Aug-2018
Name: issue_d, dtype: object
```

[71]:
```
rf_reg = RandomForestRegressor()
```

```
rf_reg.fit(train[pred_vars_2], train['int_rate'])
```

[71]:
```
RandomForestRegressor()
```

[72]:
```
reg_multi = sm.OLS(train['int_rate'], train[pred_vars_2], hasconst=False).fit()
reg_multi.summary()
```

[72]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:               int_rate   R-squared (uncentered):
0.909
Model:                           OLS   Adj. R-squared (uncentered):
0.909
Method:                Least Squares   F-statistic:
1.202e+05
Date:               Sun, 24 Oct 2021   Prob (F-statistic):
0.00
Time:                       12:00:35   Log-Likelihood:
-2.0678e+05
No. Observations:               72378   AIC:
4.136e+05
Df Residuals:                   72372   BIC:
4.136e+05
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
=======
                           coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-------
log_funded_amnt          0.2436      0.012     20.823      0.000       0.221
0.266
inq_last_6mths           1.2707      0.018     69.440      0.000       1.235
1.307
open_acc                -0.0271      0.003     -9.423      0.000      -0.033
-0.021
bc_util                  0.0492      0.001     89.244      0.000       0.048
0.050
log_last_pymnt_amnt      0.0349      0.010      3.520      0.000       0.015
0.054
duration                 0.1686      0.002    107.561      0.000       0.166
0.172
================================================================================
```

```
Omnibus:                       8168.087   Durbin-Watson:                   1.995
Prob(Omnibus):                    0.000   Jarque-Bera (JB):            12027.371
Skew:                             0.852   Prob(JB):                         0.00
Kurtosis:                         4.041   Cond. No.                         91.0
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[73]:
```
svr_reg = SVR()

svr_reg.fit(train[pred_vars_2], train['int_rate'])
```

[73]:
```
SVR()
```

[74]:
```
mlp_reg = MLPRegressor()

mlp_reg.fit(train[pred_vars_2], train['int_rate'])
```

[74]:
```
MLPRegressor()
```

[75]:
```
pls2 = PLSRegression(n_components=6, max_iter=500, scale=True, copy=True)
pls2.fit(train[pred_vars_2], train['int_rate'])
```

[75]:
```
PLSRegression(n_components=6)
```

[77]:
```
models = [reg_multi, rf_reg, svr_reg, mlp_reg, pls2]

for reg in models:

    reg_pred = reg.predict(test[pred_vars_2])

    reg_rmse = metrics.mean_squared_error(test['int_rate'], reg_pred,␣
 ↪squared=False)
    print(reg, "RMSE:", reg_rmse)
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper object at
0x000001CD0A3502B0> RMSE: 4.236275396917207
RandomForestRegressor() RMSE: 3.0699739788754488
SVR() RMSE: 4.2502699580155126
MLPRegressor() RMSE: 4.192579530830218
PLSRegression(n_components=6) RMSE: 4.199080377814592
```

Using the predictor variables that I used in lab 8all the model's RMSE used in this lab imporved. from the original pred_vars. Although the random forest model improved the most and stayed as

the best model, the most interesting was the MLP model. Using the new variables the MLP went from the worst model to being the second best model using RMSE.

[ ]: