Credit Default Analysis

Oscar Eli Vasquez

August 3, 2020

1 Predicting Credit Default

This project uses data from Lending Club, a peer to peer lending system, to predict the probability of credit default based on a person's characteristics. Machine learning algorithms used are: logistics regression and random trees from Python's sci-kit learn package. Analysis and conclusion of study is then presented on a website optimized for tablet viewing created using Javascript components.

[1]:
```
# Import dependencies
import pandas as pd
import numpy as np
from datetime import datetime
from matplotlib import pyplot as plt


import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn import linear_model, datasets
from sklearn.model_selection import train_test_split
from  sklearn.ensemble  import RandomForestClassifier
```

[2]:
```
# Import data url
url = 'Lending_Club_Stats_2015_v2.csv'

# Create dataframe
loan_df = pd.read_csv(url, low_memory = False)
```

[3]:
```
# Drop any NaNs in the dataframe
loan_df.dropna()

# Display for QC purposes
loan_df.head()
```

[3]:
```
     id  loan_amnt  term  int_rate grade sub_grade emp_length home_ownership  \
0  1085      10850    36      0.18     D        D5  10+ years       MORTGAGE
1  2406      15000    36      0.12     C        C1    7 years            OWN
```

```
2  3565       4000    36      0.11      B         B4     2 years           OWN
3  3713      35000    36      0.09      B         B2   < 1 year       MORTGAGE
4  3783      24000    36      0.08      B         B1     2 years          RENT

   annual_inc loan_status             purpose addr_state  dti  Default_Status
0     47000.0  Fully Paid    home_improvement         CA    0               0
1     97000.0  Fully Paid               house         IL    0               0
2     36000.0     Current                 car         CT    0               0
3    200000.0     Current    home_improvement         CA    0               0
4     98000.0     Current  debt_consolidation         OR    0               0
```

[4]:
```python
# Replace various symbols in emp_length column
loan_df['emp_length_clean'] = loan_df.emp_length.str.replace('+','')
loan_df['emp_length_clean'] = loan_df.emp_length_clean.str.replace('<','')
loan_df['emp_length_clean'] = loan_df.emp_length_clean.str.replace('years','')
loan_df['emp_length_clean'] = loan_df.emp_length_clean.str.replace('year','')
loan_df['emp_length_clean'] = loan_df.emp_length_clean.str.replace('n/a','0')
```

[5]:
```python
# Map a grade to a number for classification purposes later on
loan_df['grade_clean'] = loan_df['grade'].map({'A':7,'B':6,'C':5,'D':4,'E':3,'F':
 ↪2,'G':1})
```

1.1 Regression: Home Ownership

Logistic Regression is a type of classification algorithm involving a linear discriminant where the output is a probability that the given input point belongs to a certain class. In our case, that class is default or no default.

Logistical regression is used here to determine the correlation between the categorical factors of home ownership type. While none of these factors were highly correlated to default status, it is surprising that "mortgage" was least correlated to default status while "own" a home is the second least correlated. It is not surprising that "Rent" is the highest correlation to default.

[6]:
```python
# Create headers using home_ownership variables using 0 and 1 corresponding to␣
 ↪each user
home_ownership = pd.get_dummies(loan_df.home_ownership)
loan_df = loan_df.join(home_ownership)
loan_df.head()
```

[6]:
```
     id  loan_amnt  term  int_rate grade sub_grade emp_length home_ownership  \
0  1085      10850    36      0.18      D         D5  10+ years       MORTGAGE
1  2406      15000    36      0.12      C         C1    7 years            OWN
2  3565       4000    36      0.11      B         B4    2 years            OWN
3  3713      35000    36      0.09      B         B2   < 1 year       MORTGAGE
4  3783      24000    36      0.08      B         B1    2 years           RENT

   annual_inc loan_status             purpose addr_state  dti  Default_Status  \
```

```
0     47000.0  Fully Paid    home_improvement         CA    0                0
1     97000.0  Fully Paid               house         IL    0                0
2     36000.0     Current                 car         CT    0                0
3    200000.0     Current    home_improvement         CA    0                0
4     98000.0     Current  debt_consolidation         OR    0                0

  emp_length_clean  grade_clean  ANY  MORTGAGE  OWN  RENT
0               10            4    0         1    0     0
1                7            5    0         0    1     0
2                2            6    0         0    1     0
3                1            6    0         1    0     0
4                2            6    0         0    0     1
```

[7]:
```
# Define classifier to be used to estimate fit. Here the default Limited-Memory␣
↪Broyden-Fletcher-Goldfarb-Shanno algorithm is used
# The LBFGS works well for small datasets saving on computational power as it␣
↪uses an approximation of the Hessian matrix
clf = linear_model.LogisticRegression(solver='lbfgs')
```

[8]:
```
# Define variables to be tested and fill matrix of training data X using 0's and␣
↪1's corresponding to variables
X_variables_home = ['RENT', 'MORTGAGE', 'OWN', 'ANY']
X_home= loan_df[X_variables_home]
X_home = X_home.values
y_home = loan_df['Default_Status'].values

# Fit the model to training data
model_home= clf.fit(X_home,y_home)

# Return accuray of model
model_home.score(X_home,y_home)
```

[8]: 0.8318051746043055

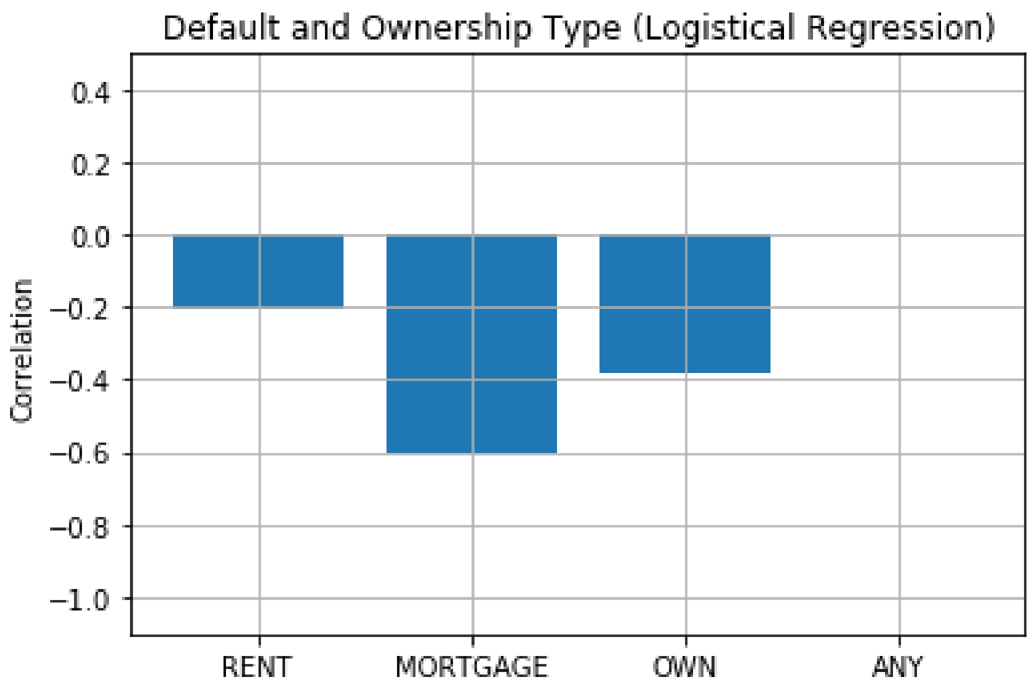
[9]:
```
# Create dataframe for results with column headers 0 and 1 relating to the␣
↪variables and correlation results respectively
homeowner_df = pd.DataFrame(list(zip(X_variables_home, model_home.coef_.T)))
homeowner_df[1] = homeowner_df[1].str.get(0)
homeowner_df.head()
```

[9]:

	0	1
0	RENT	-0.207057
1	MORTGAGE	-0.602304
2	OWN	-0.379346
3	ANY	-0.000163

[10]:

```
# Plot the data as a bar chart
h_x_axis = homeowner_df[0]
h_y_axis = homeowner_df[1]
x_axis = np.arange(len(h_x_axis))
plt.bar(x_axis, h_y_axis, align="edge")
tick_locations = [value+0.4 for value in x_axis]
plt.xticks(tick_locations, h_x_axis)

plt.title("Default and Ownership Type (Logistical Regression)")
plt.ylabel("Correlation")

plt.ylim(min(h_y_axis) - .5, max(h_y_axis) + .5)
plt.grid()

plt.show()
```



1.2 Regression: Loan Purpose

Regression was used to determine the correlation between the categorical factors of Loan Purpose. Based on the results of the test, the loan purpose with the highest correlation to default status is "Small Business" and the second highest correlation is "Renewable Energy". The loan purpose with the lowest correlation was "Wedding" and "Credit Card".

[11]:
```python
# Create headers using purpose variables using 0 and 1 corresponding to each␣
 ↪user
purpose = pd.get_dummies(loan_df.purpose)
loan_df = loan_df.join(purpose)
```

[12]:
```python
# Define classifier to be used to estimate fit. Here the default Limited-Memory␣
 ↪Broyden-Fletcher-Goldfarb-Shanno algorithm is used
# The LBFGS works well for small datasets saving on computational power as it␣
 ↪uses an approximation of the Hessian matrix
X_variables_purpose = ['debt_consolidation', 'credit_card', 'home_improvement',␣
 ↪'house','medical','other','major_purchase'
                  ␣
 ↪,'car','small_business','vacation','moving','renewable_energy','wedding','educational']
X_purpose= loan_df[X_variables_purpose]
X_purpose = X_purpose.values
y_purpose = loan_df['Default_Status'].values
model_purpose = clf.fit(X_purpose,y_purpose)
model_purpose.score(X_purpose,y_purpose)
```

[12]: 0.8318051746043055

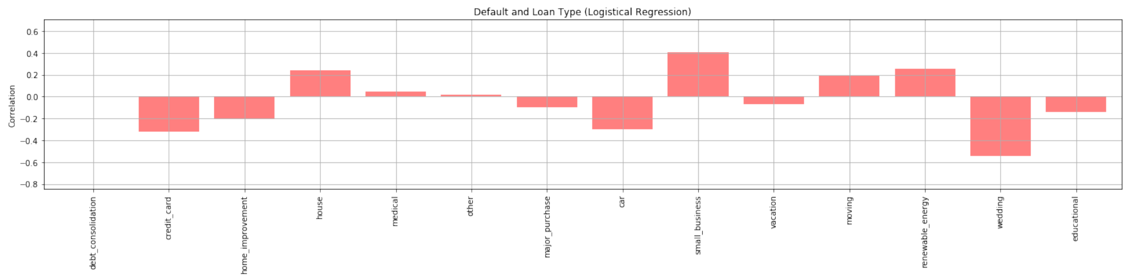
[13]:
```python
# Create dataframe for results with column headers 0 and 1 relating to the␣
 ↪variables and correlation results respectively
purpose_df = pd.DataFrame(list(zip(X_variables_purpose, model_purpose.coef_.T)))
purpose_df[1] = purpose_df[1].str.get(0)
purpose_df.head()
```

[13]:

	0	1
0	debt_consolidation	0.000266
1	credit_card	-0.322255
2	home_improvement	-0.206879
3	house	0.240535
4	medical	0.042703

[14]:
```python
# Plot the data as a bar chart

p_x_axis = purpose_df[0]
p_y_axis = purpose_df[1]

x_axis = np.arange(len(p_x_axis))
tick_locations = [value+0.4 for value in x_axis]

plt.figure(figsize = (20,5))
plt.xticks(tick_locations, p_x_axis, rotation="vertical")

plt.xlim(-0.25, len(x_axis))
plt.ylim(min(p_y_axis) - .3, max(p_y_axis) + .3)
```

```python
plt.title("Default and Loan Type (Logistical Regression)")
plt.ylabel("Correlation")

bars = plt.bar(x_axis, p_y_axis, alpha = .5, color = "r", align="edge")
plt.grid()
plt.tight_layout()

plt.show()
```



1.3 Regression: Employment

Regression was used to determine the correlation between the categorical factors of employment. The test shows that 10+ years of employment length is the least correlated to a default status. The factor with the highest correlation to default was "1 year" of experience. However, none of these factors were highly correlated to default status.

[15]:
```python
# Create headers using employment length (emp_length) variables using 0 and 1␣
↪corresponding to each user
employed = pd.get_dummies(loan_df.emp_length)
loan_df = loan_df.join(employed)
```

[16]:
```python
# Define classifier to be used to estimate fit. Here the default Limited-Memory␣
↪Broyden-Fletcher-Goldfarb-Shanno algorithm is used
# The LBFGS works well for small datasets saving on computational power as it␣
↪uses an approximation of the Hessian matrix
X_variables_employ = ['< 1 year','1 year','2 years','3 years','4 years','5␣
↪years','6 years','7 years','8 years','9 years','10+ years']
X_employ = loan_df[X_variables_employ]
y_employ = loan_df['Default_Status'].values
model_employ = clf.fit(X_employ,y_employ)
model_employ.score(X_employ,y_employ)
```

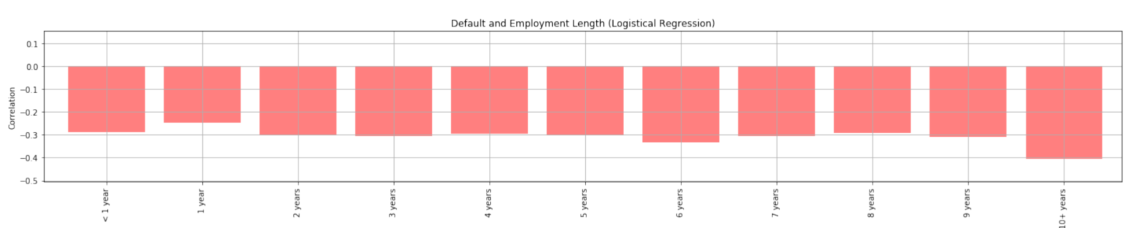
[16]: 0.8318051746043055

[17]:
```python
# Create dataframe for results with column headers 0 and 1 relating to the␣
↪variables and correlation results respectively
employ_df = pd.DataFrame(list(zip(X_employ,model_employ.coef_.T)))
employ_df[1] = employ_df[1].str.get(0)
employ_df.head()
```

[17]:

	0	1
0	< 1 year	-0.287320
1	1 year	-0.246421
2	2 years	-0.303327
3	3 years	-0.305441
4	4 years	-0.295461

[18]:
```python
# Plot the data as a bar chart

e_x_axis = employ_df[0]
e_y_axis = employ_df[1]

x_axis = np.arange(len(e_x_axis))
tick_locations = [value+0.4 for value in x_axis]

plt.figure(figsize = (20,4))
plt.xticks(tick_locations, e_x_axis, rotation="vertical")

plt.xlim(-0.25, len(x_axis))
plt.ylim(min(e_y_axis) - .1, max(e_y_axis) + .4)

plt.title("Default and Employment Length (Logistical Regression)")
plt.ylabel("Correlation")

bars = plt.bar(x_axis, e_y_axis, alpha = .5, color = "r", align="edge")
plt.grid()
plt.tight_layout()

plt.show()
```



1.4 Regression: Loan Grade

Regression was also used to determine the correlation between the categorical factors of loan grade. Lending Club assigns a grade to each loan based on credit score and a combination of several internal indicators of credit risk. This test shows that the internal model assigning a grade within lending club is doing a good job of determining risky loans. For example, the A graded loans have the lowest correlation to default status and the G rated loans have the highest correlation to default status.

[19]:
```
# Create headers using load grade variables using 0 and 1 corresponding to each␣
 ↪user
grade = pd.get_dummies(loan_df.grade)
loan_df = loan_df.join(grade)
```

[20]:
```
# Define classifier to be used to estimate fit. Here the default Limited-Memory␣
 ↪Broyden-Fletcher-Goldfarb-Shanno algorithm is used
# The LBFGS works well for small datasets saving on computational power as it␣
 ↪uses an approximation of the Hessian matrix
X_variables_grade = ['A','B','C','D','E','F','G']
X_grade = loan_df[X_variables_grade]
y_grade = loan_df['Default_Status'].values
model_grade = clf.fit(X_grade,y_grade)
model_grade.score(X_grade,y_grade)
```

[20]: 0.8318051746043055

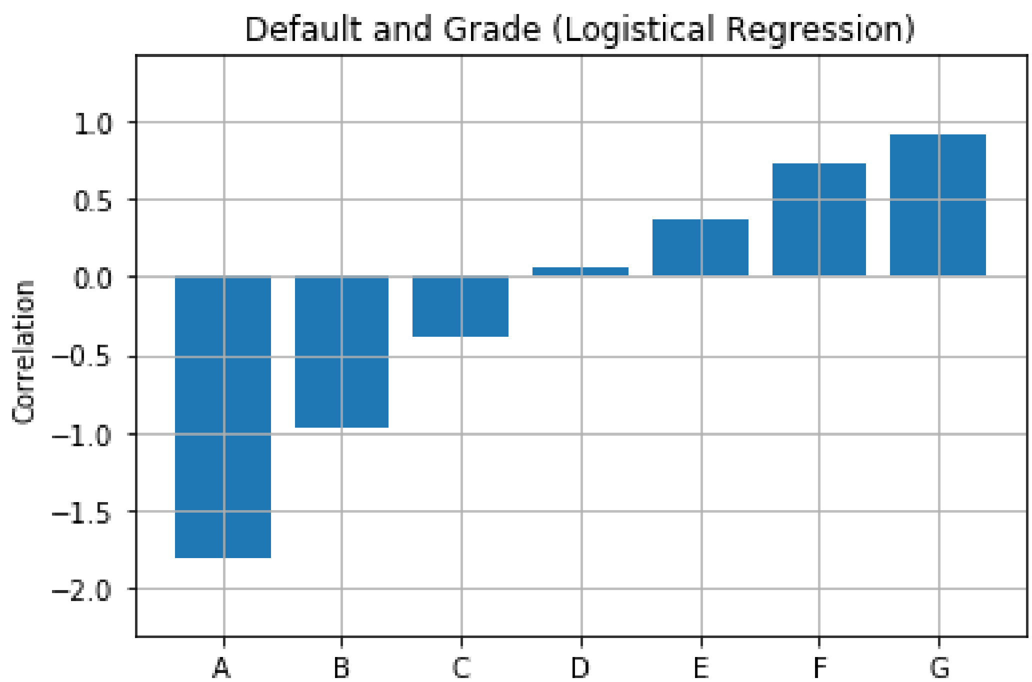
[21]:
```
# Create dataframe for results with column headers 0 and 1 relating to the␣
 ↪variables and correlation results respectively
grade_df = pd.DataFrame(list(zip(X_grade,model_grade.coef_.T)))
grade_df[1] = grade_df[1].str.get(0)
grade_df.head()
```

[21]:

	0	1
0	A	-1.802105
1	B	-0.977668
2	C	-0.393115
3	D	0.054283
4	E	0.359339

[22]:
```
# Plot the data as a bar chart

g_x_axis = grade_df[0]
g_y_axis = grade_df[1]
x_axis = np.arange(len(g_x_axis))
plt.bar(x_axis, g_y_axis, align="edge")
tick_locations = [value+0.4 for value in x_axis]
plt.xticks(tick_locations, g_x_axis)
```

```
plt.title("Default and Grade (Logistical Regression)")
plt.ylabel("Correlation")

plt.ylim(min(g_y_axis) - .5, max(g_y_axis) + .5)
plt.grid()

plt.savefig("./img/matplotlib_figures/default_and_grade.png")

plt.show()
```



1.5 Regression Summary

Following the logistical regression tests on each categorical factor we tested the accuracy of the model in predicting default status. The training and testing data scores were both .81. The overall accuracy of predictions was 82%.

[23]:
```
# Creating scratch dataframe for accuracy calculations
loan_prediction_df = pd.read_csv(url, low_memory = False)
loan_prediction_df = loan_prediction_df.
  ↪drop(["id","grade","sub_grade","emp_length","home_ownership","loan_status","purpose","addr_st
  ↪axis=1)
```

[24]:
```
# Define x and y to go into model
X = loan_prediction_df.drop("Default_Status", axis=1)
```

```
y = loan_prediction_df['Default_Status']
print(X.shape, y.shape)
```

(421095, 5) (421095,)

[25]:
```
# Input dataset to split the data. This will randomly split the lending tree␣
↪data into test and training data
# Define classifier to be used to estimate fit. Here the default Limited-Memory␣
↪Broyden-Fletcher-Goldfarb-Shanno algorithm is used
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1,␣
↪stratify=y)
```

[26]:
```
# Use the classifier to predict defaults and not defaults for entire dataset
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
print(f"First 10 Predictions:   {predictions[:10]}")
print(f"First 10 Actual labels: {y_test[:10].tolist()}")
```

```
First 10 Predictions:   [0 0 0 0 0 0 0 0 0 0]
First 10 Actual labels: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
```

[27]:
```
# Create a dataframe to store comparison between predicted and accurate results
logistic_regression_prediction_df = pd.DataFrame({"Prediction": predictions,␣
↪"Actual": y_test}).reset_index(drop=True)
logistic_regression_prediction_df.head()
```

[27]:

	Prediction	Actual
0	0	0
1	0	0
2	0	0
3	0	1
4	0	0

[28]:
```
# Scanning dataframe for Prediction and Actual matches and calculating percent␣
↪correct
lrp_correct = logistic_regression_prediction_df.
↪loc[logistic_regression_prediction_df["Actual"] ==␣
↪logistic_regression_prediction_df["Prediction"]]
correct_percentage = (lrp_correct["Actual"].count())/
↪(logistic_regression_prediction_df["Actual"].count())*100
print(correct_percentage)
```

83.1800824515075

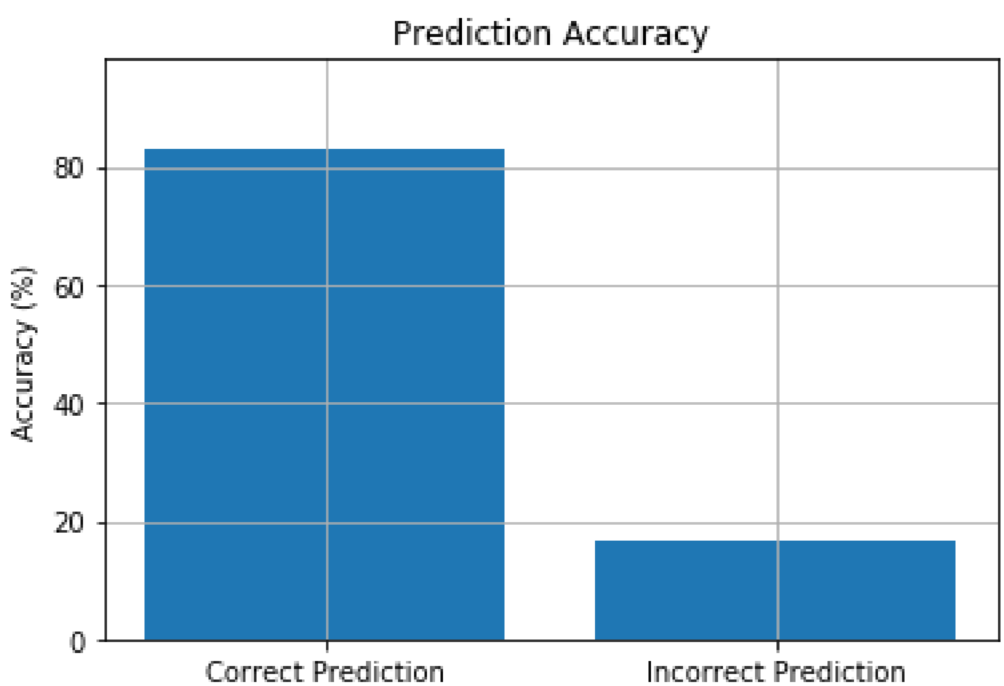
[29]:
```
# Scanning dataframe for Prediction and Actual matches and calculating percent␣
↪incorrect
```

```
lrp_incorrect = logistic_regression_prediction_df.
 ↪loc[logistic_regression_prediction_df["Actual"] !=␣
 ↪logistic_regression_prediction_df["Prediction"]]
incorrect_percentage = (lrp_incorrect["Actual"].count())/
 ↪(logistic_regression_prediction_df["Actual"].count())*100
print(incorrect_percentage)
```

16.819917548492505

[30]:

```
# Plot the data as a bar chart

bar_values = [correct_percentage, incorrect_percentage]
bar_names = ["Correct Prediction", "Incorrect Prediction"]
x_axis = np.arange(len(bar_values))
plt.bar(x_axis, bar_values, align="edge")
tick_locations = [value+0.4 for value in x_axis]
plt.xticks(tick_locations, bar_names)

plt.title("Prediction Accuracy")
plt.ylabel("Accuracy (%)")

plt.ylim(0, max(bar_values) + 15)
plt.grid()


plt.show()
```



1.6 Classification

A random forest test was also used in order to determine the correlation between various numerical features and default status. Random forests or random decision forests operate by constructing decision trees at training time and outputting the classification or regression of the individual trees. The model score after fitting was .81. After assessing the model fit a feature importance test was ran to determine which feature was best correlated. The best correlated feature was "annual_inc" and the least correlated feature was "term". This is likely because the annual income of the borrower would make it more difficult to repay a loan. The second most correlated feature was "loan_amnt" as a higher loan amount would be harder to repay.

[31]:
```
# Creating scratch dataframe for accuracy calculations
loan_classify_df = pd.read_csv(url, low_memory = False)
```

[32]:
```
# Identify target variables (whether loan defaulted or was paid) and drop any␣
↪unused charcteristics
target = loan_classify_df["Default_Status"]
target_names = ['Default','Paid']
loan_classify_df = loan_classify_df.
↪drop(["Default_Status","id","grade","sub_grade","emp_length","home_ownership","loan_status","
↪axis=1)
feature_names = loan_classify_df.columns
loan_classify_df.dropna().head()
```

[32]:

	loan_amnt	term	int_rate	annual_inc	dti
0	10850	36	0.18	47000.0	0
1	15000	36	0.12	97000.0	0
2	4000	36	0.11	36000.0	0
3	35000	36	0.09	200000.0	0
4	24000	36	0.08	98000.0	0

[33]:
```
# Split data into test and training set
X_train, X_test, y_train, y_test = train_test_split(loan_classify_df, target,␣
↪random_state=42, stratify=target)
```

[34]:
```
# Defining a classifier as rf for Random Forest using 200 estimators and output␣
↪score
rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

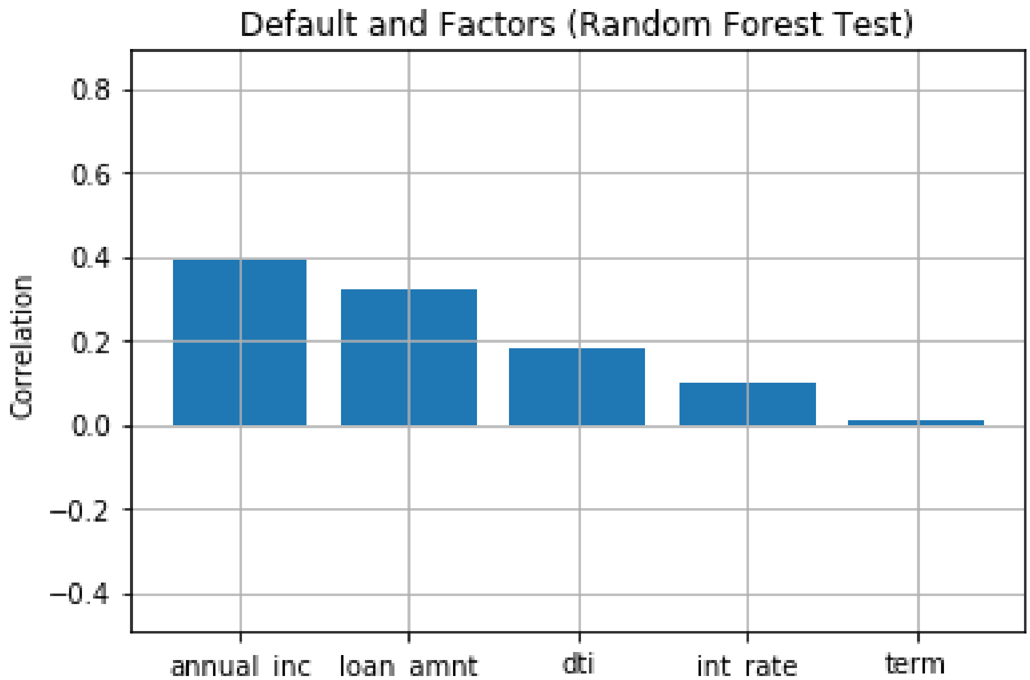
[34]: 0.8004255561677147

[35]:
```
# Create a dataframe to store results where column headers 0 and 1 attribute to␣
↪correlation and variable respectively
random_forest_df = pd.DataFrame(sorted(zip(rf.feature_importances_,␣
↪feature_names), reverse=True))
random_forest_df.head()
```

[35]:

	0	1
0	0.390084	annual_inc
1	0.320353	loan_amnt
2	0.179878	dti
3	0.099550	int_rate
4	0.010135	term

[36]:
```
# Plot the data as a bar chart

r_x_axis = random_forest_df[1]
r_y_axis = random_forest_df[0]
x_axis = np.arange(len(r_x_axis))
plt.bar(x_axis, r_y_axis, align="edge")
tick_locations = [value+0.4 for value in x_axis]
plt.xticks(tick_locations, r_x_axis)

plt.title("Default and Factors (Random Forest Test)")
plt.ylabel("Correlation")

plt.ylim(min(r_y_axis) - .5, max(r_y_axis) + .5)
plt.grid()


plt.show()
```



1.7 Summary and Conclusions

The purpose of this project was to predict the probability of credit default based on a credit owner's characteristics uaing Lending Tree data from 2015. Logistics regression and random tree analyses were used for prediction algorithms. Python's sci-kit learn library was used to run the analyses. Logistics regression determined that 10+ years of employment length is the least correlated to a default status. The factor with the highest correlation to default was "1 year" of experience. However, none of the employment lengths were highly correlated to default status. For loan purpose, the highest correlation to default status is "Small Business" with "Renewable Energy" as the second highest. The loan purposes with the lowest correlation were "Wedding" and "Credit Card". For grades, A graded loans have the lowest correlation to default status and the G rated loans have the highest correlation to default status. In fact this relationship appeared linear. Also, as far home ownership is concerned, none of the factors correlated with loan default. The random forest test showed "annual_inc" to correlate most with loan default, and the least correlated feature was the "term" of the loan.