Lending Club Mini Project with Class and Kaggle Influence :) We used a different data set that included data from 2007 to 2017.

We used the Mini Project outline in Class along with a Kaggle resource and data set to complete this mini project.

In [66]:
```
%matplotlib inline
import os
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
import datetime
import seaborn as sns
import time
import sklearn
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import joblib  # Updated import
from xgboost import XGBClassifier
```

In [67]:
```
import os

desktop_path = os.path.expanduser("~/Desktop")  # Assuming you're on a macOS or
file_name = "accepted_2007_to_2017.csv"
file_path = os.path.join(desktop_path, file_name)
```

In [72]:
```
ROOT_PATH = '../'
accepted = pd.read_csv(ROOT_PATH+'accepted_2007_to_2017.csv')
```

```
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/1156468918.p
y:2: DtypeWarning: Columns (0,18,48,58,117,128,129,130,133,134,135,138,144,14
5,146) have mixed types. Specify dtype option on import or set low_memory=Fals
e.
  accepted = pd.read_csv(ROOT_PATH+'accepted_2007_to_2017.csv')
```

In [73]:
```
accepted.shape
```

Out[73]:
```
(1646801, 150)
```

In [102…
```
summary_stats = accepted.describe(include='all')
print(summary_stats)
```

	loan_amnt	funded_amnt_inv	term	int_rate	installment	grade	
count	1,645,442	1,645,442	1645442	1,645,442	1,645,442	1645442	\
unique	NaN	NaN	2	NaN	NaN	7	
top	NaN	NaN	36	NaN	NaN	C	
freq	NaN	NaN	1181576	NaN	NaN	490304	
mean	14,729	14,698	NaN	13	439	NaN	
min	500	0	NaN	5	5	NaN	
25%	8,000	8,000	NaN	10	252	NaN	
50%	12,600	12,500	NaN	13	377	NaN	
75%	20,000	20,000	NaN	16	580	NaN	
max	40,000	40,000	NaN	31	1,720	NaN	
std	8,801	8,803	NaN	5	259	NaN	

	emp_title	emp_length	home_ownership	annual_inc	...	pct_tl_nvr_dlq	
count	1645442	1,645,442	1645442	1,645,442	...	1,575,033	\
unique	373062	NaN	3	NaN	...	NaN	
top	other	NaN	MORTGAGE	NaN	...	NaN	
freq	101787	NaN	814342	NaN	...	NaN	
mean	NaN	6	NaN	76,725	...	94	
min	NaN	0	NaN	600	...	0	
25%	NaN	3	NaN	46,176	...	91	
50%	NaN	7	NaN	65,000	...	98	
75%	NaN	10	NaN	92,000	...	100	
max	NaN	10	NaN	1,000,000	...	100	
std	NaN	4	NaN	50,938	...	9	

	percent_bc_gt_75	pub_rec_bankruptcies	tot_hi_cred_lim	
count	1,578,586	1,644,082	1,575,186	\
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	46	0	175,634	
min	0	0	0	
25%	12	0	50,327	
50%	50	0	113,577	
75%	75	0	253,851	
max	100	12	9,999,999	
std	36	0	177,635	

	total_bal_ex_mort	total_bc_limit	total_il_high_credit_limit	
count	1,595,431	1,595,431	1,575,186	\
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	50,746	21,939	42,989	
min	0	0	0	
25%	21,416	7,900	15,000	
50%	38,029	15,400	32,315	
75%	63,841	28,600	57,700	
max	3,408,095	1,105,500	1,736,064	
std	48,464	21,725	43,844	

	issue_yr	early_cr_yr	verified
count	1,645,442	1,645,417	1645442
unique	NaN	NaN	2
top	NaN	NaN	False
freq	NaN	NaN	1143284
mean	2,015	1,999	NaN
min	2,007	1,933	NaN
25%	2,014	1,995	NaN

```
50%         2,015        2,000        NaN
75%         2,016        2,004        NaN
max         2,017        2,014        NaN
std             2            8        NaN

[11 rows x 74 columns]
```

In [107…

```python
mean_value = accepted['loan_amnt'].mean()
median_value = accepted['loan_amnt'].median()
std_deviation = accepted['loan_amnt'].std()
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Standard Deviation: {std_deviation}")
```

```
Mean: 14728.875508829846
Median: 12600.0
Standard Deviation: 8800.74068683586
```

In [109…

```python
mean_value = accepted['funded_amnt_inv'].mean()
median_value = accepted['funded_amnt_inv'].median()
std_deviation = accepted['funded_amnt_inv'].std()

print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Standard Deviation: {std_deviation}")
```

```
Mean: 14698.00438462712
Median: 12500.0
Standard Deviation: 8802.944876380638
```

In [111…

```python
mean_value = df['int_rate'].mean()
median_value = df['int_rate'].median()
std_deviation = df['int_rate'].std()

print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Standard Deviation: {std_deviation}")
```

```
Mean: 13.218571489296068
Median: 12.74
Standard Deviation: 4.7042939790216876
```

In [112…

```python
mean_value = df['installment'].mean()
median_value = df['installment'].median()
std_deviation = df['installment'].std()

print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Standard Deviation: {std_deviation}")
```

```
Mean: 439.4122187629419
Median: 377.04
Standard Deviation: 259.22557031052946
```

Here I looked at the data size, shape and list out the summary statistics of the data. What the accepted.shape tells me is that the data set includes ~1.66M observations with 150 different features. I am suspecting that the data set will have missing data. I will try to see what data is missing. Additionally, I also know from a Python coding message from earlier that I have different types of data. DtypeWarning: Columns

(0,18,48,58,117,128,129,130,133,134,135,138,144,145,146) have mixed types. Specify dtype option on import or set low_memory=False. interactivity=interactivity, compiler=compiler, result=result)

In [74]:
```
accepted
```

Out[74]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rat
0	38098114	NaN	15000.0	15000.0	15000.0	60 months	12.3
1	36805548	NaN	10400.0	10400.0	10400.0	36 months	6.9
2	37842129	NaN	21425.0	21425.0	21425.0	60 months	15.5
3	37612354	NaN	12800.0	12800.0	12800.0	60 months	17.1
4	37662224	NaN	7650.0	7650.0	7650.0	36 months	13.6
...	...	...	...	...	...	...	.
1646796	66141895	NaN	14400.0	14400.0	14400.0	60 months	13.1
1646797	65673209	NaN	34050.0	34050.0	34050.0	36 months	15.4
1646798	65744272	NaN	5000.0	5000.0	5000.0	36 months	11.2
1646799	Total amount funded in policy code 1: 2087217200	NaN	NaN	NaN	NaN	NaN	Nal
1646800	Total amount funded in policy code 2: 662815446	NaN	NaN	NaN	NaN	NaN	Nal

1646801 rows × 150 columns

In [75]:
```
missing_data = accepted.isnull().sum().sort_values(ascending= False)
drop_columns = list(missing_data[missing_data > accepted.shape[0] *0.1].index)
accepted = accepted.drop(drop_columns, axis = 1)

missing_data
```

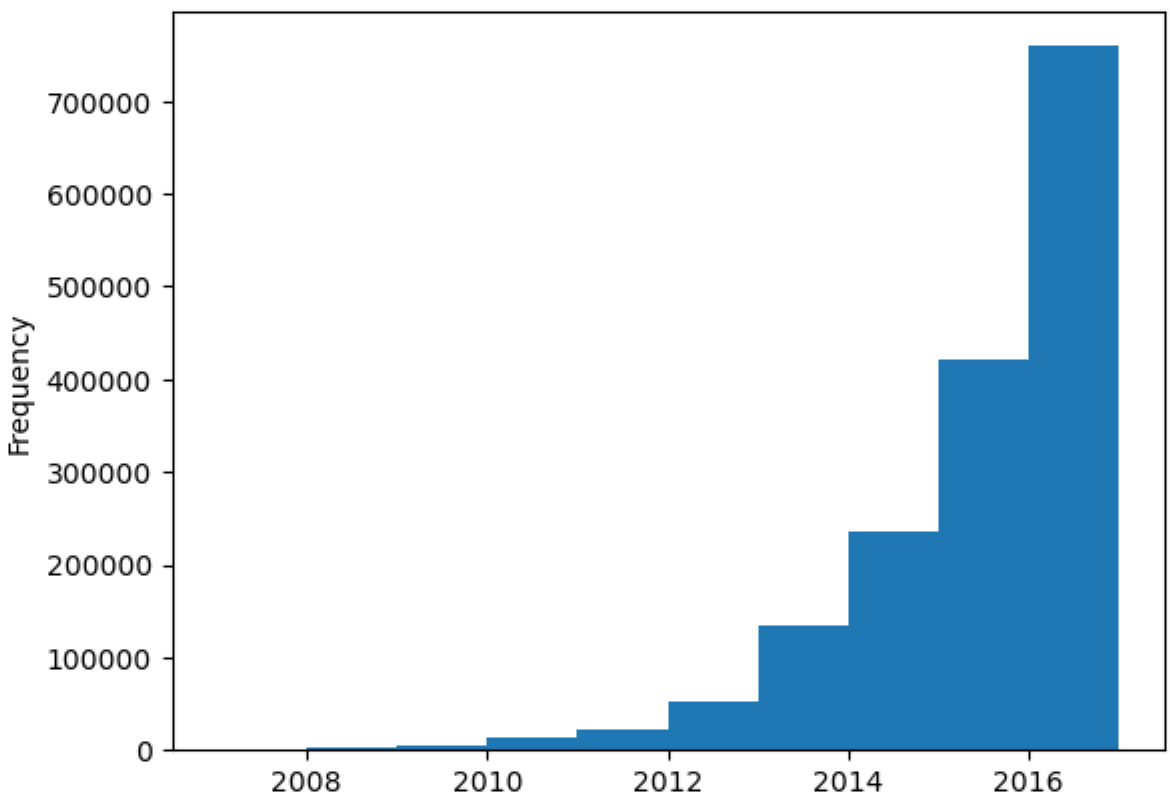
Out[75]:
```
member_id                                      1646801
orig_projected_additional_accrued_interest     1641979
hardship_end_date                              1641023
hardship_amount                                1641023
hardship_type                                  1641023
                                                ...
revol_bal                                           23
fico_range_high                                     23
fico_range_low                                      23
addr_state                                          23
id                                                   0
Length: 150, dtype: int64
```

The histogram plot below shows that many loan portfolios have changed since 2012. The code helps transfomr the issue date of the loan so that I can use this data for more analysis.

In [76]:
```
accepted.term = accepted.term.apply(str)
accepted['term'] = accepted['term'].apply(lambda x: x.strip().split(" ")[0])

accepted.issue_d = pd.to_datetime(accepted.issue_d)
accepted['issue_yr'] = accepted.issue_d.dt.year
accepted['issue_yr'].plot.hist()
```

```
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/758379839.py:
4: UserWarning: Could not infer format, so each element will be parsed individ
ually, falling back to `dateutil`. To ensure parsing is consistent and as-expe
cted, please specify a format.
  accepted.issue_d = pd.to_datetime(accepted.issue_d)
```

Out[76]: <AxesSubplot:ylabel='Frequency'>



From the data set, my plan is to not use several columns because they show a linear combintion with others and/or do not contain enough information.

In [77]:
```
accepted = accepted.drop(['title', 'funded_amnt'], axis = 1)
accepted = accepted.drop(['out_prncp_inv','total_rec_prncp','total_pymnt_inv'],
accepted = accepted.drop(['fico_range_low', 'last_fico_range_low',
                          'avg_cur_bal',
                          'addr_state', 'initial_list_status', 'pymnt_plan',
                          'application_type', 'hardship_flag', 'disbursement_met
                           'debt_settlement_flag','sub_grade',
                          'zip_code', 'id','policy_code','tax_liens', 'tax_liens
```

I would also like to transform a few variables into the date type for easier analysis later.

In [78]:
```
accepted.home_ownership = accepted.home_ownership.replace(['ANY', 'NONE','OTHER

accepted['issue_yr'] = accepted.issue_d.dt.year
accepted['earliest_cr_line'] = pd.to_datetime(accepted.earliest_cr_line)
accepted['early_cr_yr'] = accepted.earliest_cr_line.dt.year

median_year = accepted.emp_length.value_counts(ascending = False).index[0]
accepted.loc[:, 'emp_length'] = accepted.loc[:, 'emp_length'].fillna(median_yea

accepted.emp_length = accepted.emp_length.replace(['10+ years'], '10 years')
accepted.emp_length = accepted.emp_length.replace(['< 1 year'], '0 years')

accepted.emp_length = accepted.emp_length.apply(lambda x: int(str(x).split(' ')
print(accepted.emp_length.value_counts())

accepted.loc[:, 'emp_title'] = accepted.loc[:, 'emp_title'].fillna('other')
accepted.emp_title = accepted.emp_title.apply(lambda x: x.lower())
accepted.emp_title = accepted.emp_title.replace(['lpn','registered nurse', 'rn
```

```
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/1196701627.p
y:4: UserWarning: Could not infer format, so each element will be parsed indiv
idually, falling back to `dateutil`. To ensure parsing is consistent and as-ex
pected, please specify a format.
  accepted['earliest_cr_line'] = pd.to_datetime(accepted.earliest_cr_line)
emp_length
10    644759
2     148367
0     133332
3     130871
1     107680
5     101848
4      98103
6      75568
8      72664
7      70395
9      63214
Name: count, dtype: int64
```

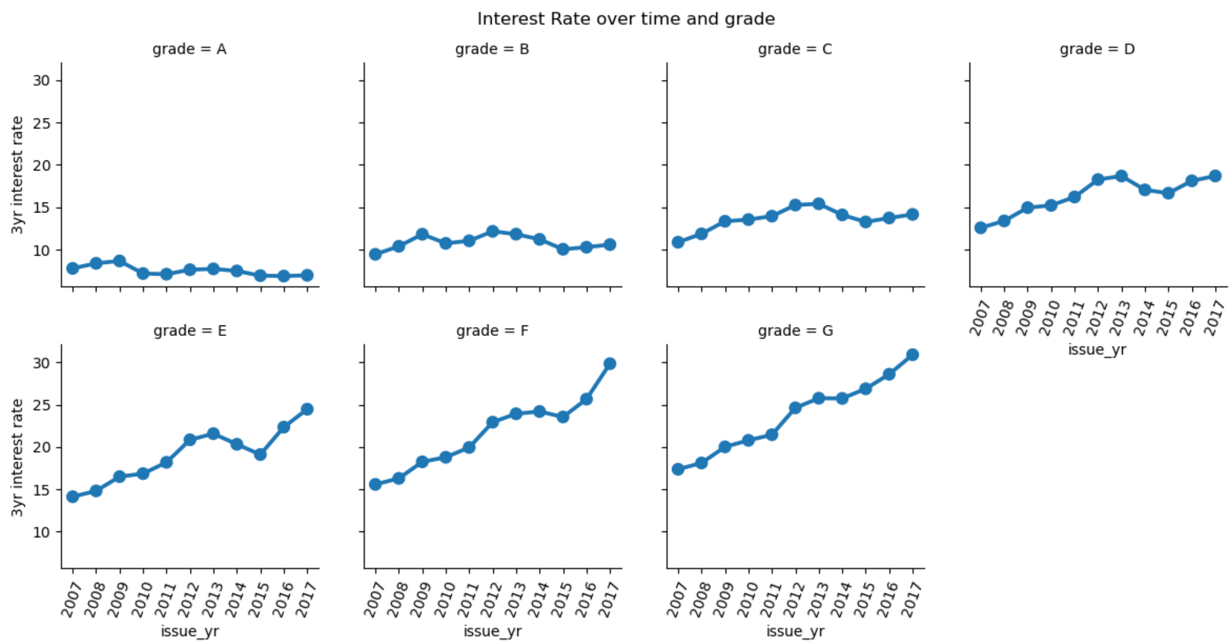
Next, we want to explore changing rates through time. Each loan normally receives a grade from G through A. Starting in 2017, grade F and G were no longer used. These codes can also help plot average interest rates over time.

In [79]:
```
rate = pd.pivot_table(accepted[accepted['term'] == '36'],index=["grade","issue_
rate.shape # 77, 1
rate = rate.reset_index()
```

In [80]:
```
g = sns.FacetGrid(rate, col = 'grade', col_wrap = 4)
g = g.map(sns.pointplot, "issue_yr", "int_rate")

labels = np.arange(2007, 2018, 1)
labels = [str(i) for i in labels]
g = g.set_xticklabels(labels, rotation=70)
g = g.set_ylabels("3yr interest rate")

plt.subplots_adjust(top=0.9)
g.fig.suptitle('Interest Rate over time and grade')
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/axisgrid.py:712: UserWarnin
g: Using the pointplot function without specifying `order` is likely to produc
e an incorrect plot.
  warnings.warn(warning)
```

Out[80]:
```
Text(0.5, 0.98, 'Interest Rate over time and grade')
```



The graphs above show that the interest rates for D,E,F,and G start increasing significantly after 2014. However, there are very few data points from 2007 to 2014, so the next thing we must do is look to see if the increase is due to an average rate and not due to a small number of observations.

In [81]:
```
# Number of observations for each grade
# to verify the variance of rates
rate_count = pd.pivot_table(accepted[accepted['term'] == '36'],index=["grade",'
rate_count = rate_count.unstack('grade')
rate_count
```

Out[81]:

	int_rate						
grade	A	B	C	D	E	F	G
issue_yr							
2007.0	78	98	141	99	100	52	35
2008.0	318	594	580	419	285	111	86
2009.0	1203	1445	1348	817	308	105	55
2010.0	2567	2805	2070	1253	336	91	34
2011.0	5579	4722	2203	1261	272	54	10
2012.0	10753	16805	9902	5088	795	103	24
2013.0	17057	40313	24693	14505	3231	608	15
2014.0	35333	53460	44042	20510	7066	1980	179
2015.0	70132	91783	77457	32740	9450	1363	248
2016.0	66862	114783	92317	36707	9932	2364	530
2017.0	52191	81609	69093	24905	7884	1386	674

What this tells us is that it is resulting from the exponential increase in loans since 2007.

Why are people borrowing? Next, we will look at why (or the purpose) loans are being taken out. As you can see from the data below, the purpose of borrorwing includes debt consolidation, credit card debt, home improvement loans, major purchases, etc.). Debt consolidation was the leading reason for loans and education was the last on the list.

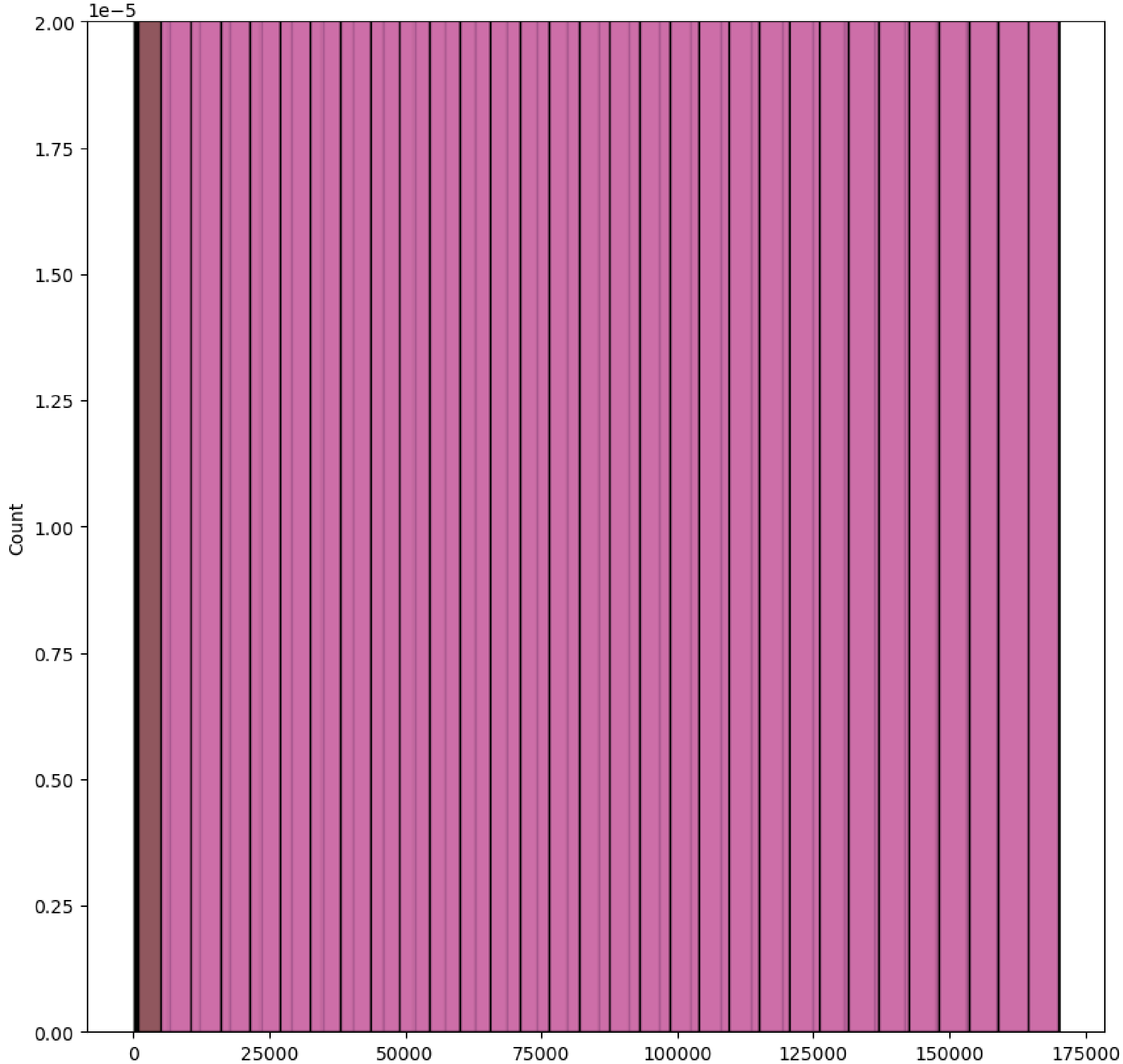
In [82]:
```
accepted.purpose.value_counts().sort_values(ascending=False)
```

Out[82]:
```
purpose
debt_consolidation    955783
credit_card           363962
home_improvement      109031
other                  93576
major_purchase         35596
medical                18901
small_business         18613
car                    17641
moving                 11388
vacation               11152
house                   7268
wedding                 2350
renewable_energy        1094
educational              423
Name: count, dtype: int64
```

In [83]:
```
incomeVerified = accepted[accepted['verification_status'] != 'Not Verified'].dı
incomeVerified = incomeVerified[['grade','annual_inc']]
quantile_low = incomeVerified['annual_inc'].min()
quantile_high = incomeVerified['annual_inc'].quantile(0.95)
filtered = incomeVerified[(incomeVerified['annual_inc'] > quantile_low) & (incc
```

In [89]:
```
grade_list = filtered['grade'].unique()
plt.figure(figsize=(10, 10))
for i in range(len(grade_list)):
    data = filtered[filtered['grade'] == grade_list[i]]['annual_inc'].values
    sns.histplot(data, bins=30)

plt.ylim(ymax=0.00002)
```

Out[89]:
```
(0.0, 2e-05)
```

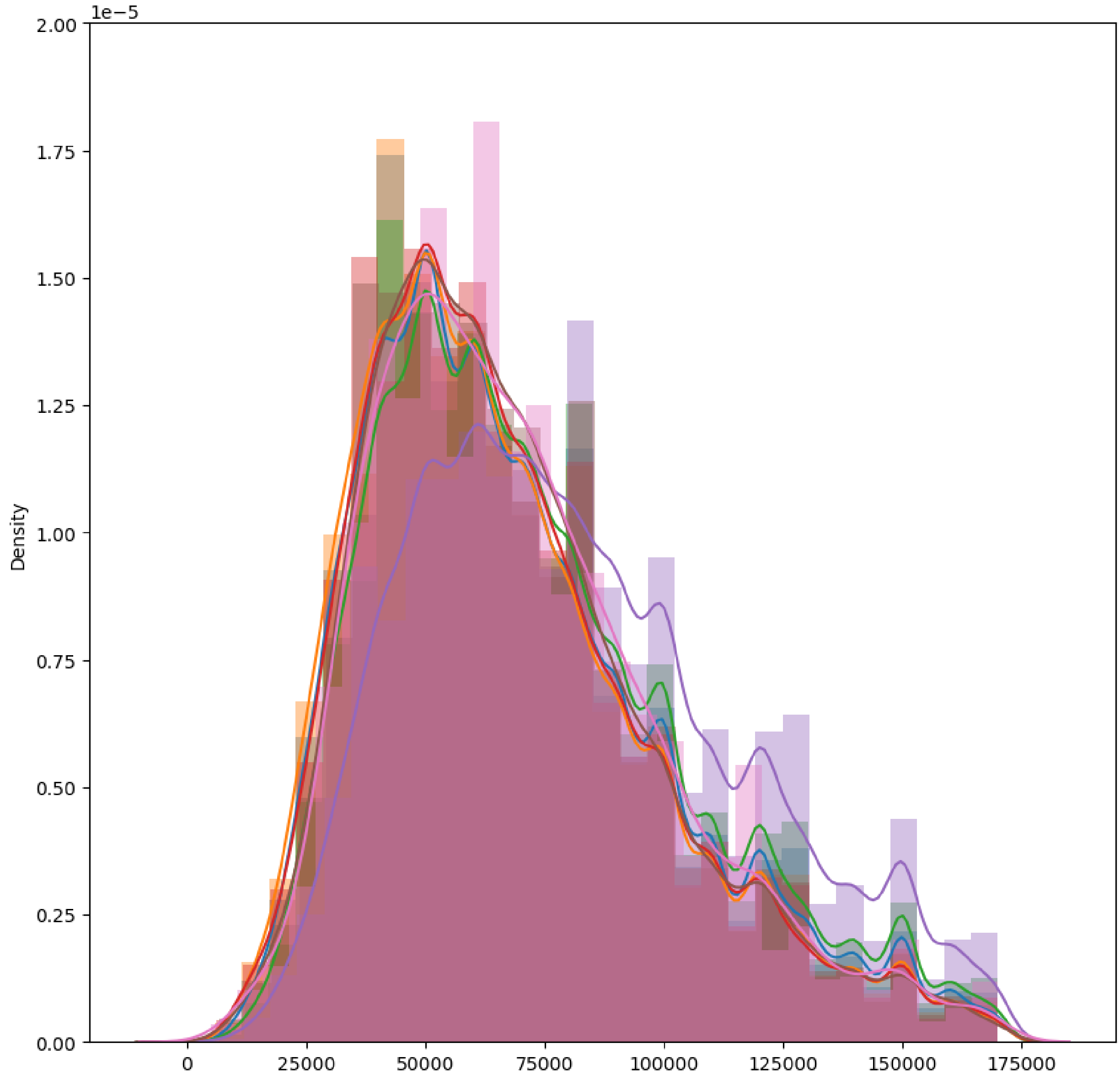


In [94]:
```
grade_list = filtered['grade'].unique()
plt.figure(figsize=(10,10))
for i in range(len(grade_list)):
    data = filtered[filtered['grade'] == grade_list[i]]['annual_inc'].values
    sns.distplot(data, bins = 30)

plt.ylim(ymax = 0.00002)
```

```
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
```

```
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
/var/folders/n5/l10z0lr57hdfl08j5wcd1gcc0000gn/T/ipykernel_85555/2724925822.p
y:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data, bins = 30)
```

Out[94]:
```
(0.0, 2e-05)
```



Here we looked at loans where income had been verified. The data variable verification status included verified, source verified and not verified. The graph above shows that there are differences in income levels. Incomes greater thand $100khaveahigherdensitythandthoselessthan$100k. This graph also insinuates that the each grade has a different median income. The next line of code will dive deeper into this and see what the median income is for each grade.

In [95]:
```
income_median = pd.pivot_table(filtered, values = 'annual_inc', index = 'grade
income_median
```

Out[95]:

grade	annual_inc
A	75600.0
B	65000.0
C	62000.0
D	60000.0
E	60320.0
F	62400.0
G	65000.0

Next, we would like to know more about what types of professionals are taking out loans. Becasue the data set is very large, this makes it difficult to build a reliable distribution of annual incomes. But we will do the following: -Filtering for loans where the reported income is less than 1 million USD -Filtering for loans where Debt-to-Income ratio is less than 100 percent. If it is greater -than or equal to 100, I wonder why we would have made such loans in the first place. I could have been more careful by capping dti value at 100. -Changing employment years into numeric -Filling unknown values for home-ownership as Rent - Standardizing the values of employment title (emp_title)

In [96]:
```
leq1mil = accepted['annual_inc'] <= 1e6
accepted = accepted[leq1mil]
accepted = accepted[accepted.dti < 100.0]
```

Additionally, because people will most likely lie on their incomes when their income is low, we can filter out for data if:

Income is lower than $70,000 but has been verified by Lending Club Income is higher than$ 70,000 but lower than $120,000 Income is higher than$120,000 but has been verified by Lending Club The choice of limit of $70,000 and$120,000 is arbitrary to filter out loans where income levels seem unrealistic.

In [97]:
```
pd.options.display.float_format = '{:,.0f}'.format
salary_limit = 7e4

emp_annual_all = accepted.loc[((accepted['annual_inc'] >= 1.2e5) & (accepted['v
                              | ((accepted['annual_inc'] >= salary_limit) & (ac
                              | ((accepted['annual_inc'] < salary_limit) & (acc
                              ['emp_title','annual_inc']].groupby('emp_title')


summ_inc = emp_annual_all.agg(['min','mean','median','max', 'count'])
summ_inc.columns = summ_inc.columns.levels[1]
summ_inc = summ_inc.sort_values(by = ['count','min'], ascending = False)
```

In [98]:
```
# Here we filter for professions with more than 500 observations
summ_inc = summ_inc[summ_inc['count'] >= 500].sort_values(by = ['count','min'],
summ_inc
```

Out[98]:

emp_title	min	mean	median	max	count
other	3,000	53,822	48,000	920,000	69250
nurse	10,900	83,810	80,000	320,000	21692
teacher	9,000	74,160	75,000	367,500	16332
manager	10,000	84,635	82,000	1,000,000	16300
owner	5,000	92,850	85,000	1,000,000	9643
...	...	...	...	...	...
inspector	20,000	80,989	80,000	200,000	511
investigator	30,000	86,423	85,000	200,000	508
dental hygienist	22,600	76,496	75,000	150,000	506
sergeant	29,000	90,779	90,000	211,000	504
occupational therapist	30,000	85,929	85,000	220,000	500

145 rows × 5 columns

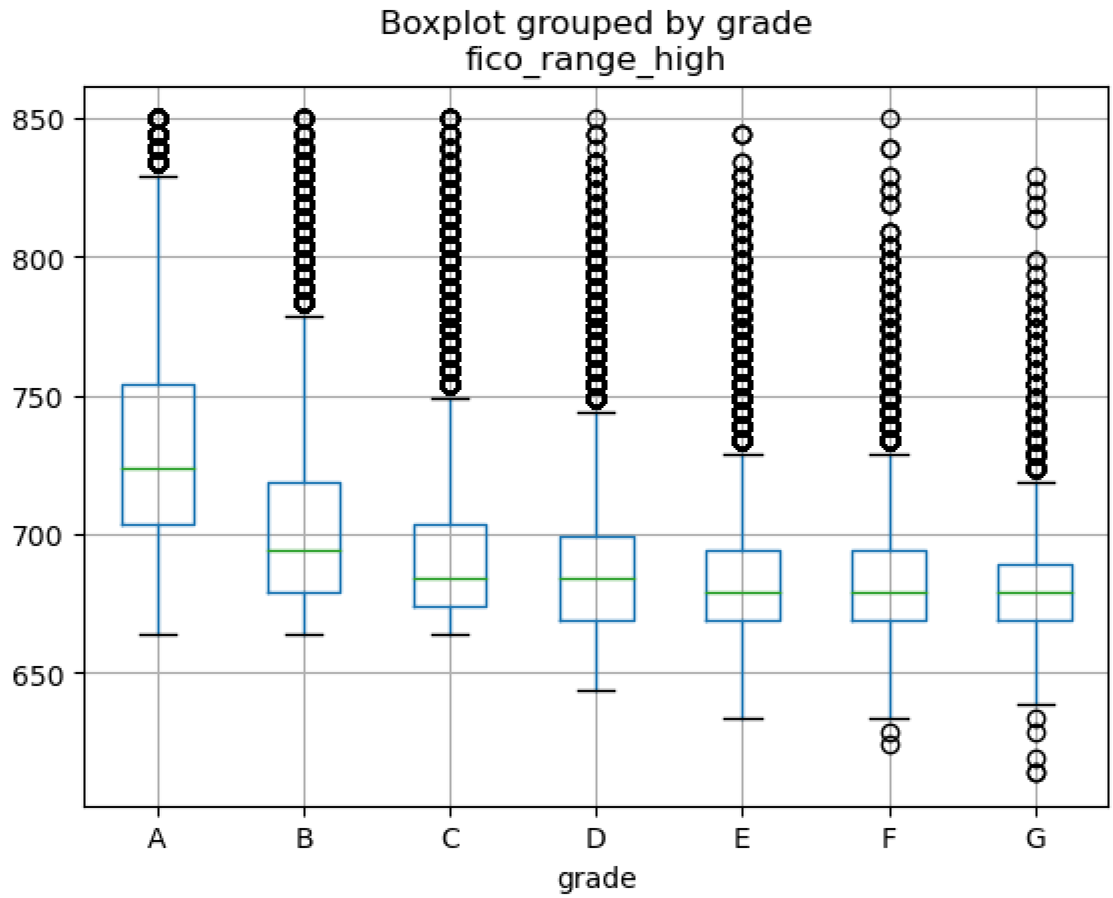
What is interesting about this data is the following: 1) The dataset does have a large diversity in the professions and income level of borrowers. Personally, what we found most interesting is that: a. A teacher and nurse, despite being regarded as respectable professions in some regions in the world, are the 2 most common professions on Lending Club. Their minimum salary is only around $12,000, which is lower than US' Poverty Level for individual. b. A police officer has a min$ 32,000 c. Many people with annual income higher than $400,000 and even$1,000,000 still use Lending Club as a way to borrow cheaply. I can dive down into the purpose of the loans later, but that may not offer much benefit for data modeling. d. Some jobs traditionally associated with high income can have very low income levels, such as attorneys, directors, or engineers. All of these salaries were verified by Lending Club. It's likely that these salaries are missing information such as 0 or more in the salary.

In [99]:

```
accepted.boxplot(by = 'grade', column = 'fico_range_high')
```

Out[99]:

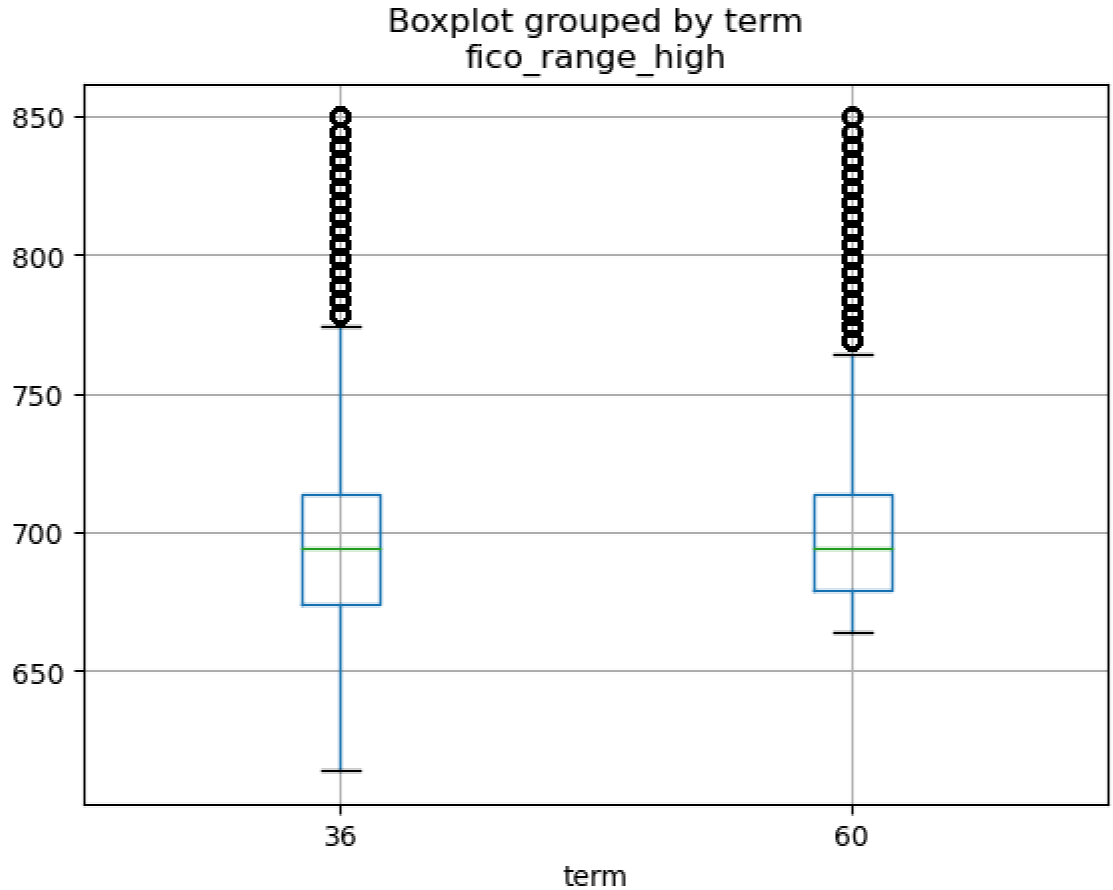
```
<AxesSubplot:title={'center':'fico_range_high'}, xlabel='grade'>
```



Looking at FICO range and grade, the graph above we can see that income distribution accross loan grades is similar to the data we looked at prior, however grade A is showing as slightly higher in this box plot schematic. It would be interesting to show FICO score changes over time.

In [114…
```
accepted.boxplot(by = 'term', column = 'fico_range_high')
```

Out[114]:
```
<AxesSubplot:title={'center':'fico_range_high'}, xlabel='term'>
```

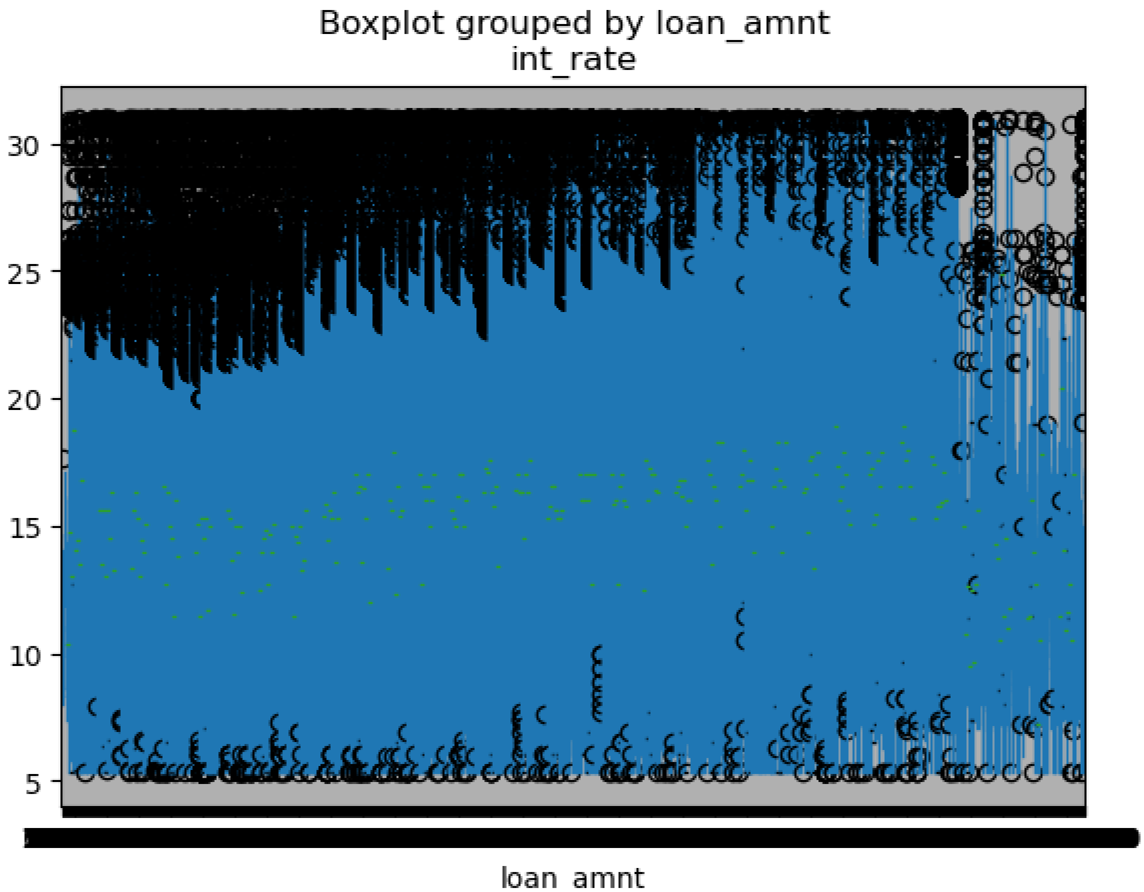


In [117…

```
accepted.boxplot(by = 'loan_amnt', column = 'int_rate')
```

Out[117]:

```
<AxesSubplot:title={'center':'int_rate'}, xlabel='loan_amnt'>
```

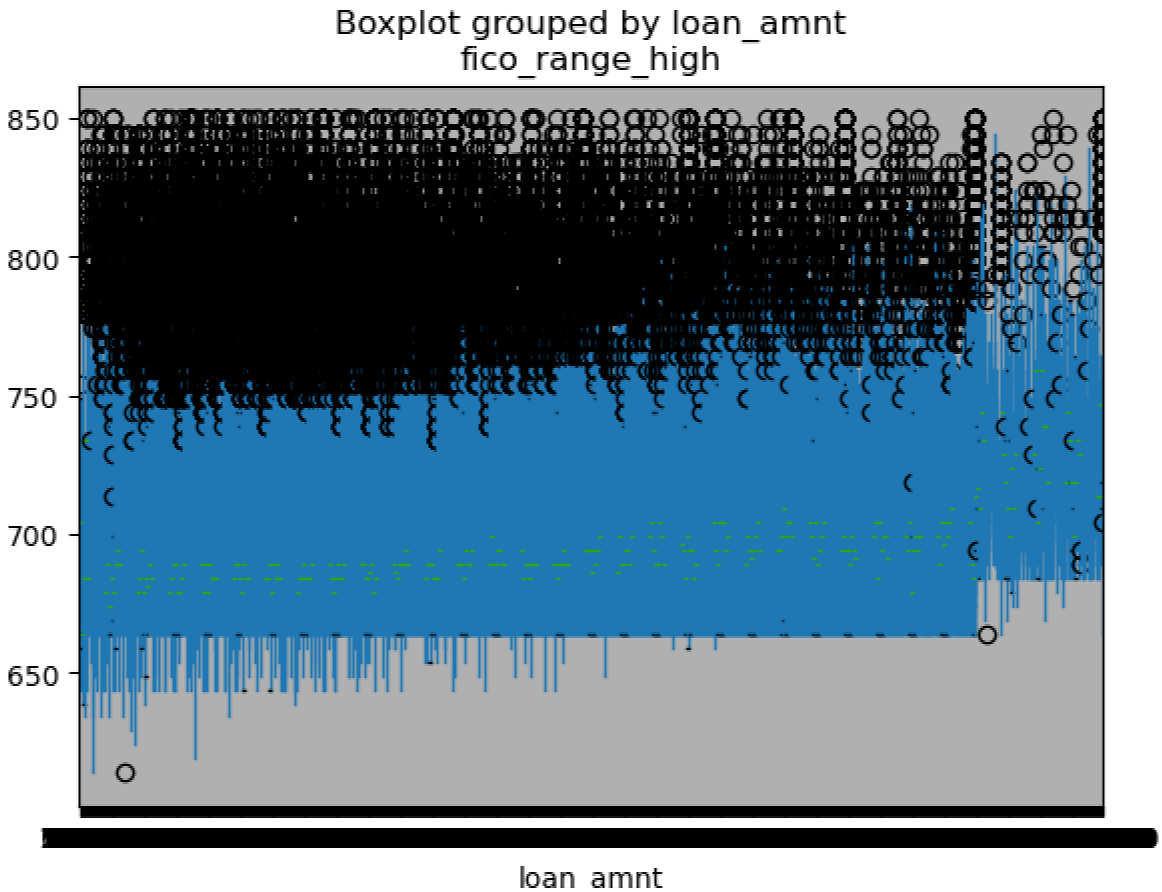


In [118…

```
accepted.boxplot(by = 'loan_amnt', column = 'fico_range_high')
```

Out[118]:

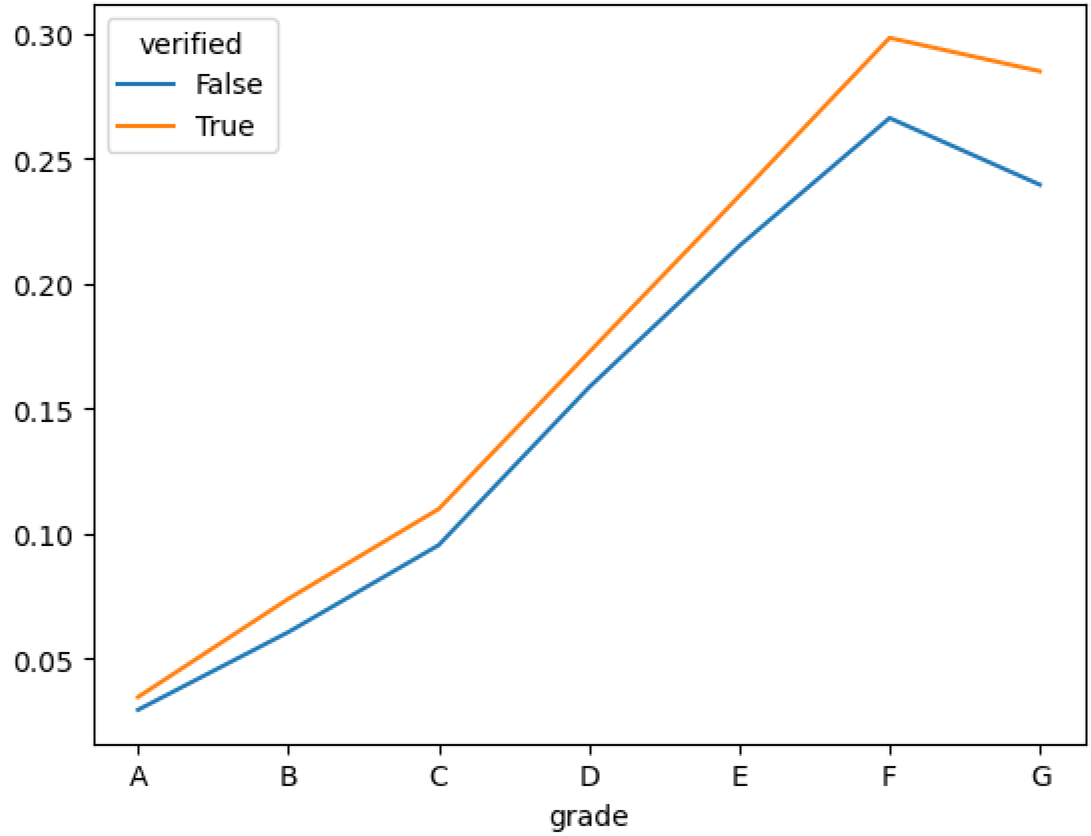
```
<AxesSubplot:title={'center':'fico_range_high'}, xlabel='loan_amnt'>
```



In [100…

```
accepted['verified'] = accepted['verification_status'] == 'Verified'
grade_yr_loanamnt = pd.pivot_table(accepted,index=["grade","verified"], values=

grade_yr_loanamnt_default = pd.pivot_table(accepted[(accepted.loan_status == '(
                                           index=["grade","verified"], values=[

grade_yr_loanamnt_default.columns = ['Charged_off']

loan_verified = pd.merge(grade_yr_loanamnt, grade_yr_loanamnt_default, left_inc
loan_verified['chargeoff_rate']  = loan_verified['Charged_off'] /  loan_verifie

loan_verified_unstack = loan_verified.unstack("verified")
verified_chargedoff = loan_verified_unstack['chargeoff_rate']
verified_chargedoff.plot()
```

Out[100]: <AxesSubplot:xlabel='grade'>



If we assumem that Lending Club verifies a borrowers income only when it is high, this graph above shows that this is not true. The graph shows that the charge off rate changes almost linearly from grade A (the highest grade) through to grade G. The charge-off rate for grade F-G is approximately 25%.

In [119…

```
accepted.loan_status.value_counts()
```

Out[119]:

```
loan_status
Current                                                787907
Fully Paid                                             646694
Charged Off                                            168043
Late (31-120 days)                                      23742
In Grace Period                                         10461
Late (16-30 days)                                        5781
Does not meet the credit policy. Status:Fully Paid       1983
Does not meet the credit policy. Status:Charged Off       761
Default                                                    70
Name: count, dtype: int64
```

In [ ]:

```
This completes the Mini project.
```