In [202]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
```

Project Overview

- You are assigned the task of assisting Lending Tree in there decision of approving a client for a loan.
- You are to create a model that will help Lending Tree in making a decision based on the features of a client
- This will assist Lending Tree to approve loans that will less likely to be Charges off and also if the loan will most likely be "Paid Off"

Target Class

- loan_status - This is the Classification that will be assigned to a loan potential for a clients application
- The model will tell Lending Tree is this loan is approved what status will be assigned to the loan: Fully Paid(1), Charged OFF(0)

Loading Data

- Will load data set from local path
- Because this is a large data file this will avoid all issues with git handling files

In [203]:
```
file_path = "/Users/berkatbhatti/Downloads/TF_2_Notebooks_and_Data/DATA/lending_club_loan_two.csv"
```

Feature Description

- This document displays all the details of each column feature

In [204]:
```
pd.read_csv("resources/lending_club_info.csv")
```

Out[204]:

	LoanStatNew	Description
0	loan_amnt	The listed amount of the loan applied for by t...
1	term	The number of payments on the loan. Values are...
2	int_rate	Interest Rate on the loan
3	installment	The monthly payment owed by the borrower if th...
4	grade	LC assigned loan grade
5	sub_grade	LC assigned loan subgrade
6	emp_title	The job title supplied by the Borrower when ap...
7	emp_length	Employment length in years. Possible values ar...
8	home_ownership	The home ownership status provided by the borr...
9	annual_inc	The self-reported annual income provided by th...
10	verification_status	Indicates if income was verified by LC, not ve...
11	issue_d	The month which the loan was funded
12	loan_status	Current status of the loan
13	purpose	A category provided by the borrower for the lo...
14	title	The loan title provided by the borrower
15	zip_code	The first 3 numbers of the zip code provided b...
16	addr_state	The state provided by the borrower in the loan...
17	dti	A ratio calculated using the borrower’s total ...
18	earliest_cr_line	The month the borrower's earliest reported cre...
19	open_acc	The number of open credit lines in the borrowe...
20	pub_rec	Number of derogatory public records
21	revol_bal	Total credit revolving balance
22	revol_util	Revolving line utilization rate, or the amount...
23	total_acc	The total number of credit lines currently in ...
24	initial_list_status	The initial listing status of the loan. Possib...
25	application_type	Indicates whether the loan is an individual ap...
26	mort_acc	Number of mortgage accounts.
27	pub_rec_bankruptcies	Number of public record bankruptcies

In [205]:
```
df = pd.read_csv(file_path)
```

Evaluating the data

- appears that we have some data missing in a few columns but we will address them shortly
- There are alot of clients in this data set so our model should be able to get pretty good predictions with the model generated

In [206]:
```
df.info()
```

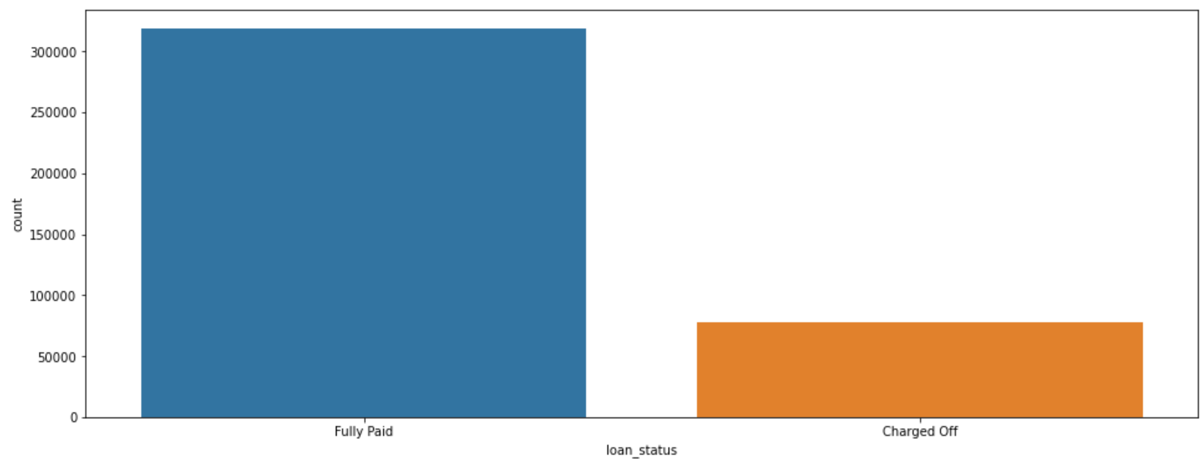
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

Target Class Analysis

- Loan status
- This is chosen to assist Lending Tree predict is a client will Pay off their loans or if they will be charged off
- Below visualization tell us that the the data set is not evenly distributed
- Since Lending tree will be more concerned with loan Charge Offs we will need a model that will have a pretty good prediction for Charge Offs, not to say that Fully Paid customers are not valuable but Lending Tree would want to know a charge off potential before assigning a Loan to a client.

In [207]:
```
plt.figure(figsize=(16,6))
sns.countplot(df["loan_status"])
```

Out[207]: <matplotlib.axes._subplots.AxesSubplot at 0x7faf85007340>

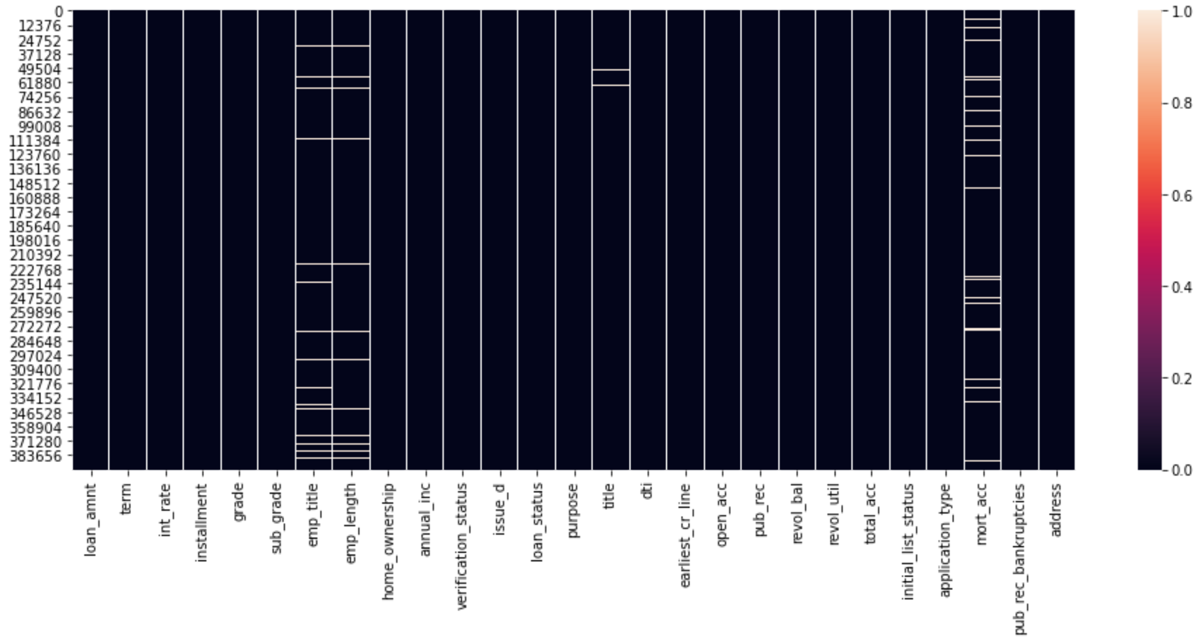


Missing Data

- Lets look at whats missing and see if we can find a way to impute these values
- There may be some cases where we will need to drop these values is they are not significant to the target
- Since we do have a large data set of clients there is room to lose a few rows
- emptitle, emp length, title, and mort account appear to be missing values .
- Will find means of updating these features for better use.

In [208]:
```
plt.figure(figsize=(16,6))
sns.heatmap(df.isnull())
```

Out[208]: <matplotlib.axes._subplots.AxesSubplot at 0x7fafc52dfcd0>



In [209]:
```
df["emp_title"].nunique()
```

Out[209]: 173105

There are a large number of unique emp_titles

- We will drop this column since it is not a strong determinant of the loan status of a client
- Also since there are so many we cannot one hot encode these values

In [210]:
```
df.drop("emp_title", axis=1, inplace=True)
```

Looking at emp_length

- There are some values missing here so lets evaluate if there is a way to replace these values
- lets refactor this column so that we get the numeric values for each client
- Lets create a dictionary to replace the values with the numeric values we want

In [211]:
```
df["emp_length"].isnull().sum()
```

Out[211]: 18301

In [212]:
```
df["emp_length"].nunique()
```

Out[212]: 11

In [213]:
```
emp_length = list(df["emp_length"].unique())
```

In [214]:
```
emp_length
```

Out[214]:
```
['10+ years',
 '4 years',
 '< 1 year',
 '6 years',
 '9 years',
 '2 years',
 '3 years',
 '8 years',
 '7 years',
 '5 years',
 '1 year',
 nan]
```

In [215]:
```
num_Length = [10,4,0,6,9,2,3,8,7,5,1,np.nan]
```

In [216]:
```
num_dictionaty = dict(zip(emp_length, num_Length))
```

In [217]:
```
num_dictionaty
```

Out[217]:
```
{'10+ years': 10,
 '4 years': 4,
 '< 1 year': 0,
 '6 years': 6,
 '9 years': 9,
 '2 years': 2,
 '3 years': 3,
 '8 years': 8,
 '7 years': 7,
 '5 years': 5,
 '1 year': 1,
 nan: nan}
```

Applying Dictionary to the Data Column

- We will replace the original values of the emp_length with the dictionary values

In [218]:
```
df["emp_length"] = df["emp_length"].apply(lambda x: num_dictionaty[x])
```

In [219]:
```
df["emp_length"].isnull().sum()
```

Out[219]:
```
18301
```

Replacing the missing values

- Lets use viaualization to get a potential average value for the emp_length

In [220]:
```
df["term"].nunique()
```

Out[220]: 2

In [221]:
```
df.head(4)
```

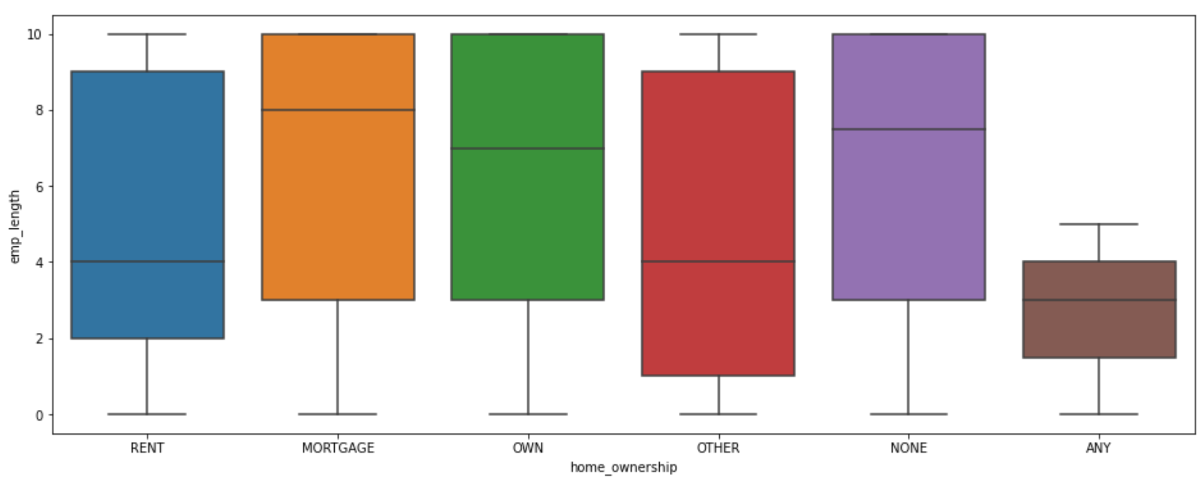
Out[221]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	anr
0	10000.0	36 months	11.44	329.48	B	B4	10.0	RENT	1
1	8000.0	36 months	11.99	265.68	B	B5	4.0	MORTGAGE	
2	15600.0	36 months	10.49	506.97	B	B3	0.0	RENT	
3	7200.0	36 months	6.49	220.65	A	A2	6.0	RENT	

4 rows × 26 columns

In [223]:
```
plt.figure(figsize=(16,6))
sns.boxplot(x = df["home_ownership"], y = df["emp_length"])
```

Out[223]: <matplotlib.axes._subplots.AxesSubplot at 0x7fafb59d20d0>



Creating a function to replace missing values

- Since home ownership has no missing values
- we will replace the missing emp_length with the average of each type of home ownership

In [22]:
```
def replaceMissingValues(cols):
    home = cols[0]
    length = cols[1]
    if pd.isnull(length):
        if home == "RENT":
            return 4.0
        elif home == "MORTGAGE":
            return 8.0
        elif home == "OWN":
            return 7.0
        elif home == "OTHER":
            return 4.0
        elif home == "NONE":
            return 7.0
        else:
            return 3.0
    else:
        return length
```

In [23]:
```
df[["home_ownership","emp_length"]]
```

Out[23]:

	home_ownership	emp_length
0	RENT	10.0
1	MORTGAGE	4.0
2	RENT	0.0
3	RENT	6.0
4	MORTGAGE	9.0
...	...	...
396025	RENT	2.0
396026	MORTGAGE	5.0
396027	RENT	10.0
396028	MORTGAGE	10.0
396029	RENT	10.0

396030 rows × 2 columns

In [24]:
```
df["emp_length"] = df[["home_ownership","emp_length",]].apply(replaceMissingValues, axis=1)
```

No missing values

- Our function worked for this case
- Since we did not have to drop the column we can use emp length in training our model

In [25]:
```
df["emp_length"].isnull().sum()
```

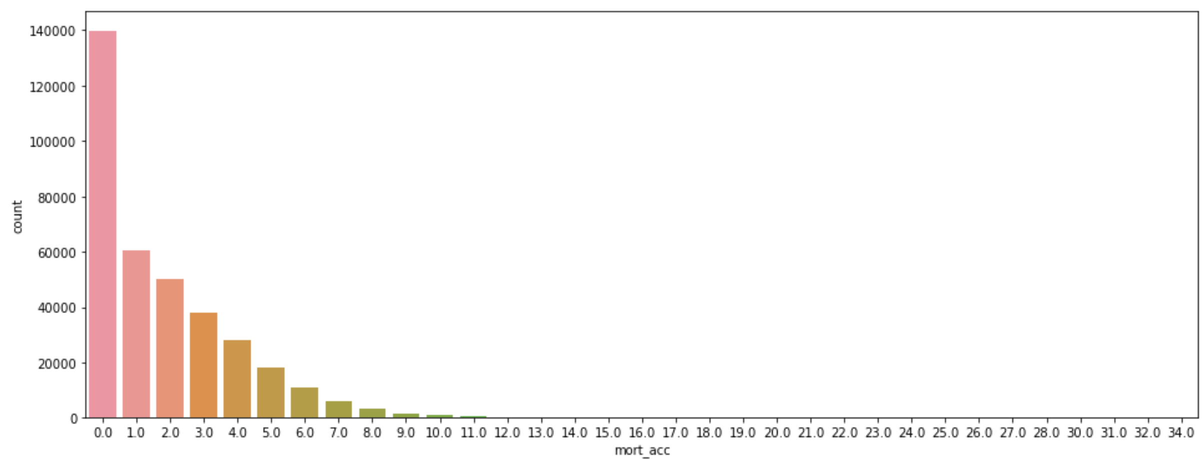
Out[25]: 0

Mortgage Account

- There is a good portion of the clients that have no Mortgage accounts

In [26]:
```
plt.figure(figsize=(16,6))
sns.countplot(df["mort_acc"])
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0a5c04af0>



Replacing Data

- We will use the same imputing process to get the average for the mortgage accounts
- This time lets take the average of the mort acc values

In [27]:
```
df.head(5)
```

Out[27]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	anı
0	10000.0	36 months	11.44	329.48	B	B4	10.0	RENT	1
1	8000.0	36 months	11.99	265.68	B	B5	4.0	MORTGAGE	
2	15600.0	36 months	10.49	506.97	B	B3	0.0	RENT	
3	7200.0	36 months	6.49	220.65	A	A2	6.0	RENT	
4	24375.0	60 months	17.27	609.33	C	C5	9.0	MORTGAGE	

5 rows × 26 columns

In [28]:
```
df["mort_acc"].mean()
```

Out[28]: 1.8139908160844138

In [29]:
```
def imputeMean(mortAcc):
    if pd.isnull(mortAcc):
        return df["mort_acc"].mean()
    else:
        return mortAcc
```

In [30]:
```
df["mort_acc"] = df["mort_acc"].apply (lambda x: imputeMean(x))
```

In [31]:
```
df["mort_acc"].isnull().sum()
```

Out[31]: 0

Title

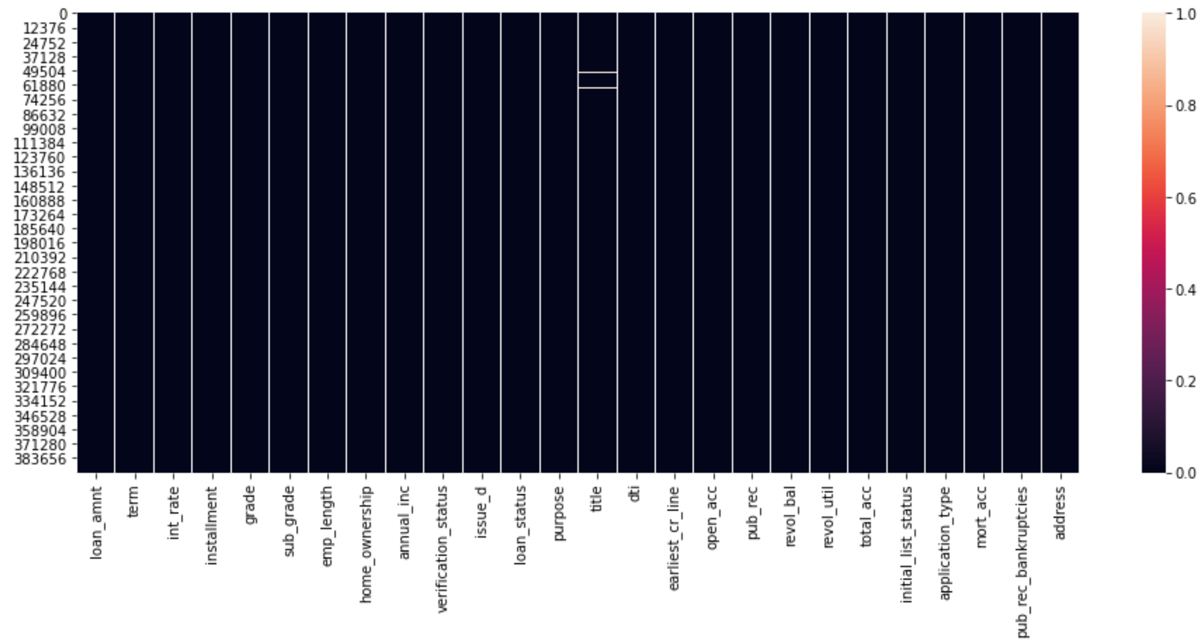
- Since there is only a few values missing here
- We will drop the rows that contains the missing values

In [32]:
```
df["title"].isnull().sum()
```

Out[32]: 1755

In [33]:
```
plt.figure(figsize=(16,6))
sns.heatmap(df.isnull())
```

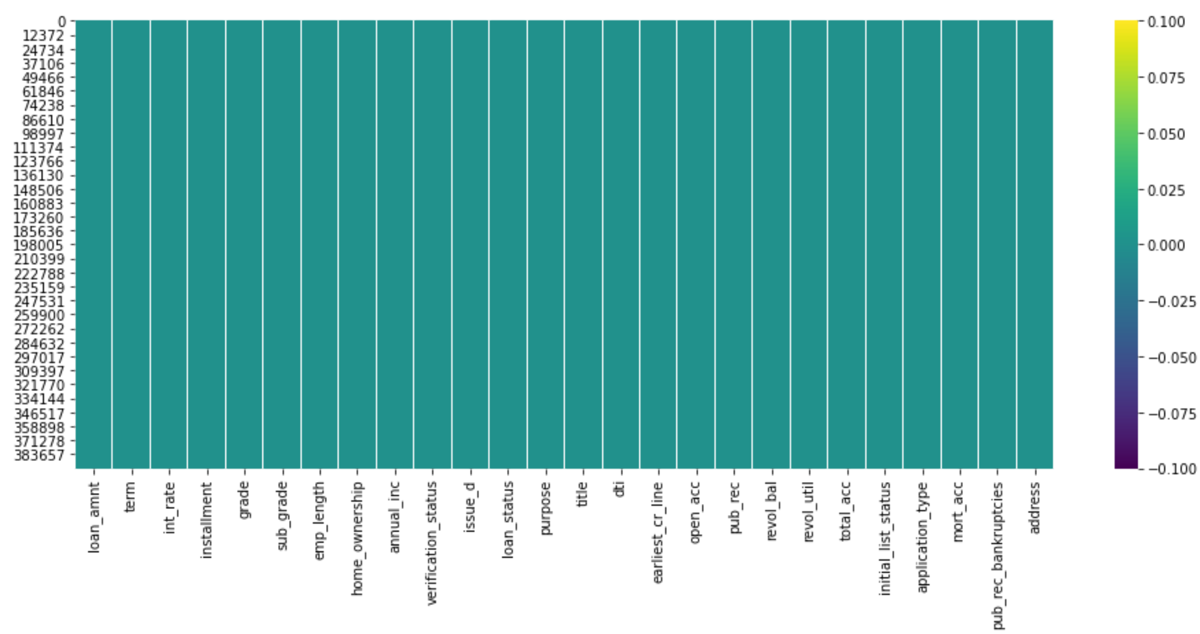
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0a4c86760>



In [34]:
```
df.dropna(inplace=True)
```

In [35]:
```
plt.figure(figsize=(16,6))
sns.heatmap(df.isnull(), cmap="viridis")
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0a4ca9ee0>

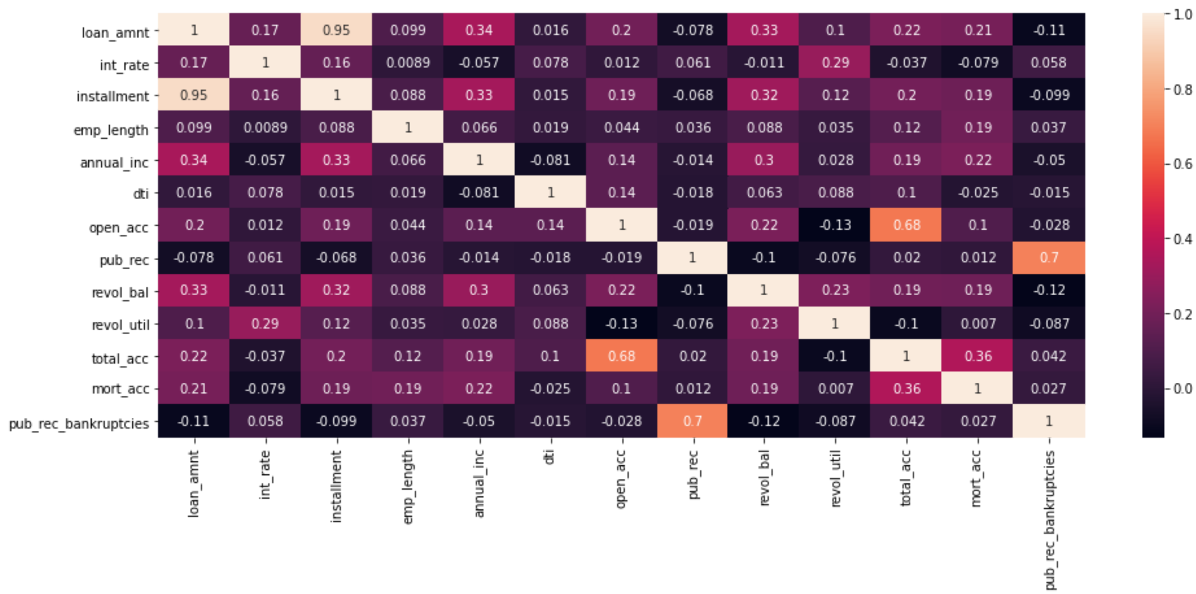


Visualization

- We will dive deep into visualization with the DataFrame we have

In [36]:
```
plt.figure(figsize=(16,6))
sns.heatmap(df.corr(), annot=True)
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0c4076520>



In [37]:
```
df.head(4)
```

Out[37]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	anr
0	10000.0	36 months	11.44	329.48	B	B4	10.0	RENT	1
1	8000.0	36 months	11.99	265.68	B	B5	4.0	MORTGAGE	
2	15600.0	36 months	10.49	506.97	B	B3	0.0	RENT	
3	7200.0	36 months	6.49	220.65	A	A2	6.0	RENT	

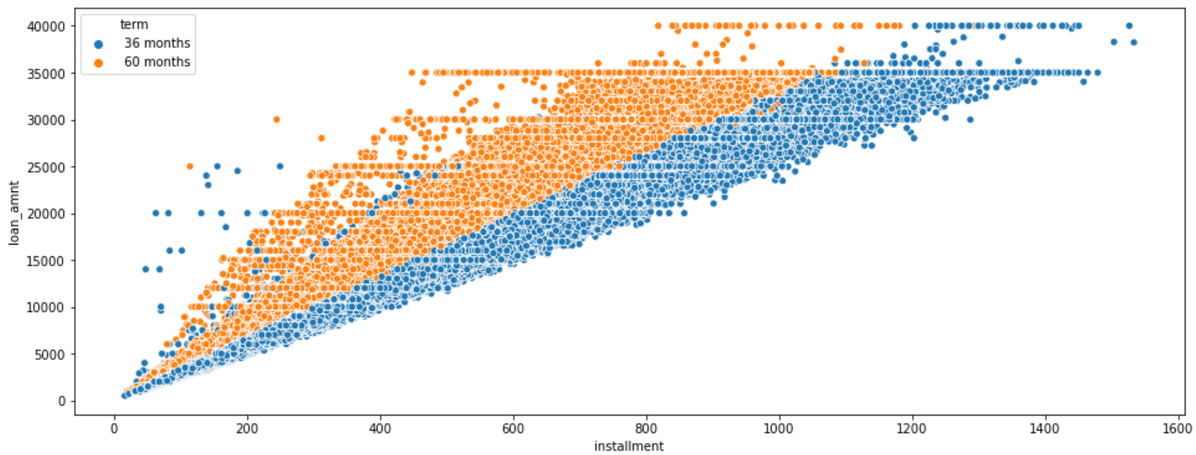
4 rows × 26 columns

Installment and loan amount

- There is a strong coorelation between loan amount and installment
- We can also see there is almost a direct indicator that larger the loan the longer the term

In [38]:
```
plt.figure(figsize=(16,6))
sns.scatterplot(x = df["installment"], y = df["loan_amnt"], hue=df["term"])
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0c1dea400>

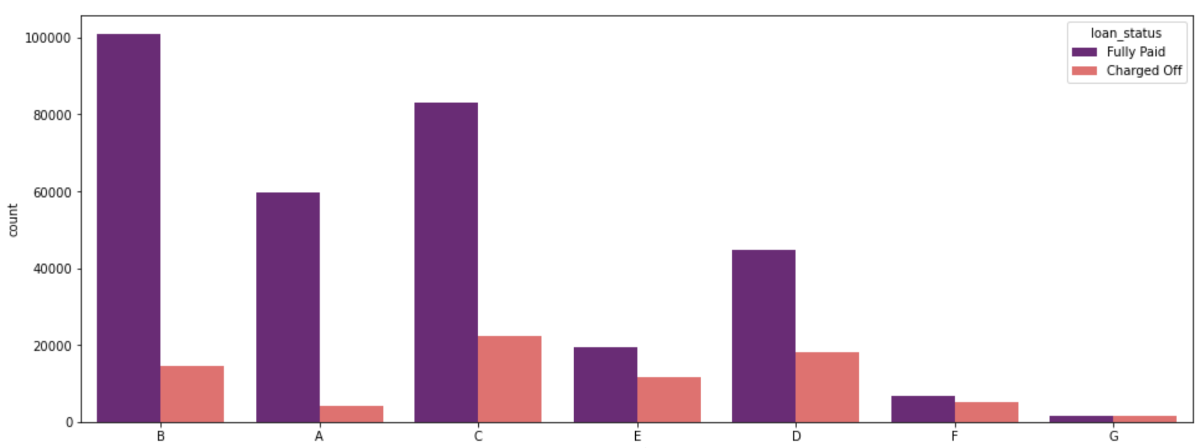


Taking a look at the Grades and loan status

- Appears the E and F grade loans are the ones to pay close attention to
- Seems to be there is a close correlation to Paid off and Charged Off

In [39]:
```
plt.figure(figsize=(16,6))
sns.countplot(x = df["grade"], hue=df["loan_status"], palette="magma")
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0c3cd9760>

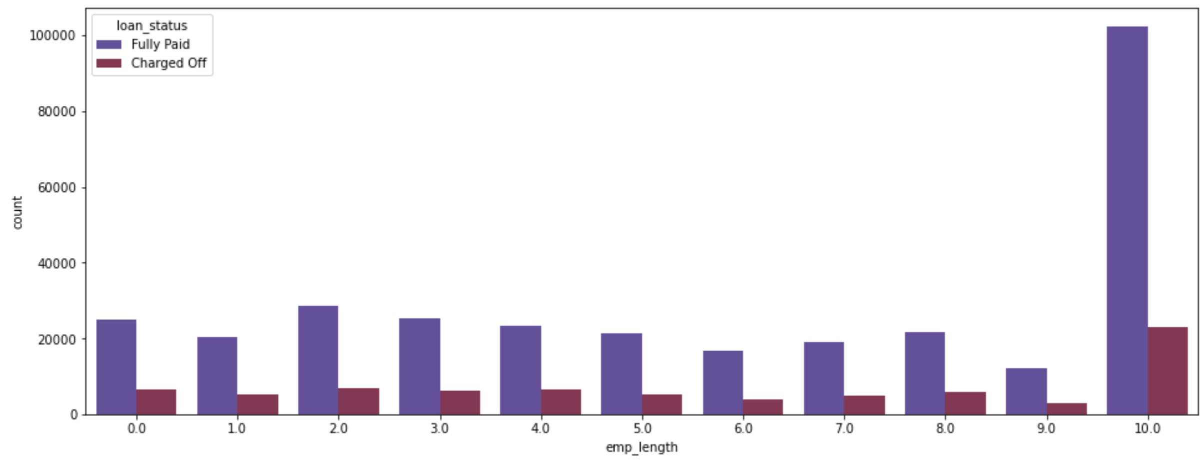


Employment length and Loan status

- Appears there is no coorelation to emp length and loan Status

In [40]:
```
plt.figure(figsize=(16,6))
sns.countplot(x = df["emp_length"], hue=df["loan_status"], palette="twilight")
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb008affeb0>



Converting Values to numeric

- Will need to convert to be read by the model

Term

- Will use list comprehension to convert these values

In [41]:
```
df["term"] = [int(x.split(" ")[1]) for x in df["term"]]
```

Grade

- We will use pandas one hot encoding processing for converting the grades

In [42]:
```
grades = pd.get_dummies(df["grade"], drop_first=True)
```

In [43]:
```
df = pd.concat([df, grades], axis=1)
```

In [44]:
```
df.drop("grade", axis=1, inplace=True)
```

Sub Grade

- We will drop this column since we already have the grade
- Since there are so my sub grades it will be too much to try to one hot encode the values

In [45]:
```
df.drop("sub_grade", axis=1, inplace = True)
```

Home owhership

- We will one hot encode these value

In [46]:
```
home_o = pd.get_dummies(df["home_ownership"], drop_first=True)
```

In [47]:
```
df = pd.concat([df, home_o], axis=1)
```

In [48]:
```
df.drop("home_ownership", axis=1, inplace=True)
```

In [49]:
```
df.head(5)
```

Out[49]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	verification_status	issue_d	lo
0	10000.0	36	11.44	329.48	10.0	117000.0	Not Verified	Jan-2015	
1	8000.0	36	11.99	265.68	4.0	65000.0	Not Verified	Jan-2015	
2	15600.0	36	10.49	506.97	0.0	43057.0	Source Verified	Jan-2015	
3	7200.0	36	6.49	220.65	6.0	54000.0	Not Verified	Nov-2014	
4	24375.0	60	17.27	609.33	9.0	55000.0	Verified	Apr-2013	Cl

5 rows × 34 columns

Verificaiton Status

In [50]:
```
veri = pd.get_dummies(df["verification_status"], drop_first=True)
```

In [51]:
```
df = pd.concat([df, veri], axis=1)
```

In [52]:
```
df.drop("verification_status", axis=1, inplace=True)
```

In [53]:
```
df.head(5)
```

Out[53]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	issue_d	loan_status	
0	10000.0	36	11.44	329.48	10.0	117000.0	Jan-2015	Fully Paid	
1	8000.0	36	11.99	265.68	4.0	65000.0	Jan-2015	Fully Paid	debt_con:
2	15600.0	36	10.49	506.97	0.0	43057.0	Jan-2015	Fully Paid	cr
3	7200.0	36	6.49	220.65	6.0	54000.0	Nov-2014	Fully Paid	cr
4	24375.0	60	17.27	609.33	9.0	55000.0	Apr-2013	Charged Off	cr

5 rows × 35 columns

Issue date

- Will use date time to get the month and year for issue_date

In [54]:
```
df["issue_d"] = [pd.to_datetime(x) for x in df["issue_d"]]
```

In [55]:
```
df["issue_month"] = df["issue_d"].apply(lambda x: x.month)
```

In [56]:
```
df["issue_year"] = df["issue_d"].apply(lambda x: x.year)
```

In [57]:
```
df["issue_day"] = df["issue_d"].apply(lambda x: x.day)
```

In [58]:
```
df.head(5)
```

Out[58]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	issue_d	loan_status	
0	10000.0	36	11.44	329.48	10.0	117000.0	2015-01-01	Fully Paid	
1	8000.0	36	11.99	265.68	4.0	65000.0	2015-01-01	Fully Paid	debt_con:
2	15600.0	36	10.49	506.97	0.0	43057.0	2015-01-01	Fully Paid	cr
3	7200.0	36	6.49	220.65	6.0	54000.0	2014-11-01	Fully Paid	cr
4	24375.0	60	17.27	609.33	9.0	55000.0	2013-04-01	Charged Off	cr

5 rows × 38 columns

In [59]:
```
df.drop("issue_d", axis=1, inplace=True)
```

In [60]:
```
df.head(2)
```

Out[60]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	loan_status	purpose
0	10000.0	36	11.44	329.48	10.0	117000.0	Fully Paid	vacation
1	8000.0	36	11.99	265.68	4.0	65000.0	Fully Paid	debt_consolidation

2 rows × 37 columns

Target | loan Status

In [61]:
```
df["loan_status"] = pd.get_dummies(df["loan_status"], drop_first=True)
```

In [62]:
```
df.head(4)
```

Out[62]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	loan_status	purpose
0	10000.0	36	11.44	329.48	10.0	117000.0	1	vacation
1	8000.0	36	11.99	265.68	4.0	65000.0	1	debt_consolidation
2	15600.0	36	10.49	506.97	0.0	43057.0	1	credit_card
3	7200.0	36	6.49	220.65	6.0	54000.0	1	credit_card

4 rows × 37 columns

Purpose

- one hot encoding the differet purpose for the loans

In [63]:
```
purp = pd.get_dummies(df["purpose"], drop_first=True)
```

In [64]:
```
df = pd.concat([df, purp], axis=1)
```

In [65]:
```
df.drop("purpose", axis=1, inplace=True)
```

In [66]:
```
df.head(4)
```

Out[66]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	loan_status	title	d
0	10000.0	36	11.44	329.48	10.0	117000.0	1	Vacation	26.2
1	8000.0	36	11.99	265.68	4.0	65000.0	1	Debt consolidation	22.0
2	15600.0	36	10.49	506.97	0.0	43057.0	1	Credit card refinancing	12.7
3	7200.0	36	6.49	220.65	6.0	54000.0	1	Credit card refinancing	2.6

4 rows × 49 columns

Title

- Dropping since we have so many value
- With so many different job title is would be very difficult to one hot encode with our expanding the features of our data dramatically.

In [67]:
```
df["title"].nunique()
```

Out[67]: 48472

In [68]:
```
df.drop("title", axis=1, inplace=True)
```

Earliest cr Line

In [69]:
```
df["earliest_cr_line"] = pd.to_datetime(df["earliest_cr_line"])
```

In [70]:
```
df["earliest_cr_line_month"] = df["earliest_cr_line"].apply(lambda x: x.month)
```

In [71]:
```
df["earliest_cr_line_year"] = df["earliest_cr_line"].apply(lambda x: x.year)
```

In [72]:
```
df.drop("earliest_cr_line", inplace=True, axis =1)
```

In [73]:
```
df.head()
```

Out[73]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	loan_status	dti	open_acc
0	10000.0	36	11.44	329.48	10.0	117000.0	1	26.24	16.0
1	8000.0	36	11.99	265.68	4.0	65000.0	1	22.05	17.0
2	15600.0	36	10.49	506.97	0.0	43057.0	1	12.79	13.0
3	7200.0	36	6.49	220.65	6.0	54000.0	1	2.60	6.0
4	24375.0	60	17.27	609.33	9.0	55000.0	0	33.95	13.0

5 rows × 49 columns

Values Check

- Since we have converted and one hot encoded a lot of values
- let make sure we did not leave out any features that need to be converted
- Appears we missed a few so lets settle up with those

In [74]:
```
df.select_dtypes(exclude="int")
```

Out[74]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	loan_status	dti	open_acc
0	10000.0	11.44	329.48	10.0	117000.0	1	26.24	16.0
1	8000.0	11.99	265.68	4.0	65000.0	1	22.05	17.0
2	15600.0	10.49	506.97	0.0	43057.0	1	12.79	13.0
3	7200.0	6.49	220.65	6.0	54000.0	1	2.60	6.0
4	24375.0	17.27	609.33	9.0	55000.0	0	33.95	13.0
...	...	...	...	...	...	...	...	...
396025	10000.0	10.99	217.38	2.0	40000.0	1	15.63	6.0
396026	21000.0	12.29	700.42	5.0	110000.0	1	21.45	6.0
396027	5000.0	9.99	161.32	10.0	56500.0	1	17.56	15.0
396028	21000.0	15.31	503.02	10.0	64000.0	1	15.88	9.0
396029	2000.0	13.61	67.98	10.0	42996.0	1	8.32	3.0

393465 rows × 43 columns

In [75]:
```
df.select_dtypes(exclude="float")
```

Out[75]:

	term	loan_status	initial_list_status	application_type	address	B	C	D
0	36	1	w	INDIVIDUAL	0174 Michelle Gateway\r\nMendozaberg, OK 22690	1	0	0
1	36	1	f	INDIVIDUAL	1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113	1	0	0
2	36	1	f	INDIVIDUAL	87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113	1	0	0
3	36	1	f	INDIVIDUAL	823 Reid Ford\r\nDelacruzside, MA 00813	0	0	0
4	60	0	f	INDIVIDUAL	679 Luna Roads\r\nGreggshire, VA 11650	0	1	0
...	...	...	...	...	...	...	...	...
396025	60	1	w	INDIVIDUAL	12951 Williams Crossing\r\nJohnnyville, DC 30723	1	0	0
396026	36	1	f	INDIVIDUAL	0114 Fowler Field Suite 028\r\nRachelborough, ...	0	1	0
396027	36	1	f	INDIVIDUAL	953 Matthew Points Suite 414\r\nReedfort, NY 7...	1	0	0
396028	60	1	f	INDIVIDUAL	7843 Blake Freeway Apt. 229\r\nNew Michael, FL...	0	1	0
396029	36	1	f	INDIVIDUAL	787 Michelle Causeway\r\nBriannaton, AR 48052	0	1	0

393465 rows × 36 columns

Applicaiton Type

In [76]:
```
app_type = pd.get_dummies(df["application_type"], drop_first=True )
```

In [77]:
```
df = pd.concat([df, app_type], axis=1)
```

In [78]:
```
df.drop("application_type", axis=1, inplace=True)
```

Initial list Status

- one hot encoding

In [79]:
```
df["initial_list_status"] = pd.get_dummies(df["initial_list_status"], dr
op_first=True)
```

Getting zip codes

- Since we cannot use the full address lets usr the zip code
- Since these are numeric values

In [80]:
```
df["address"] = [int(x.split(" ")[-1]) for x in df["address"]]
```

In [81]:
```
df.select_dtypes(include="float")
```

Out[81]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	dti	open_acc	pub_rec	rev
0	10000.0	11.44	329.48	10.0	117000.0	26.24	16.0	0.0	36
1	8000.0	11.99	265.68	4.0	65000.0	22.05	17.0	0.0	20
2	15600.0	10.49	506.97	0.0	43057.0	12.79	13.0	0.0	11
3	7200.0	6.49	220.65	6.0	54000.0	2.60	6.0	0.0	5
4	24375.0	17.27	609.33	9.0	55000.0	33.95	13.0	0.0	24
...	...	...	...	...	...	...	...	...	
396025	10000.0	10.99	217.38	2.0	40000.0	15.63	6.0	0.0	1
396026	21000.0	12.29	700.42	5.0	110000.0	21.45	6.0	0.0	43
396027	5000.0	9.99	161.32	10.0	56500.0	17.56	15.0	0.0	32
396028	21000.0	15.31	503.02	10.0	64000.0	15.88	9.0	0.0	15
396029	2000.0	13.61	67.98	10.0	42996.0	8.32	3.0	0.0	4

393465 rows × 13 columns

Dropping issue_day feature | mentioned below

In [101]:
```
df.drop("issue_day", axis=1, inplace=True)
```

Appears that our data is prepared

- We have all numeric and values that the model can read
- lets set the data and begin to train the model below

In [175]:
```
from sklearn.linear_model import LogisticRegression
```

In [176]:
```
model = LogisticRegression(max_iter= 5000)
```

Training testing and Splitting data

- 80 training and 20 testing

In [177]:
```
from sklearn.model_selection import train_test_split
```

In [178]:
```
X = df.drop("loan_status",axis=1)
y = df["loan_status"]
```

In [179]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20
, random_state=42)
```

Normalize the data

- will use the MinMaxScalar to normalize the data

In [180]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [181]:
```
scalar = MinMaxScaler()
```

In [182]:
```
X_train = scalar.fit_transform(X_train)
```

In [183]:
```
X_test = scalar.transform(X_test)
```

Fitting Data to model

- Because the data set is so large the Max Iterations for the fitting for the Logistic model must be expanded

In [184]:
```
model.fit(X_train,y_train)
```

Out[184]:
```
LogisticRegression(max_iter=5000)
```

Making predicitons

- making prediciton on the the testing values
- Because the model was not trained on these values it a perfect test for the classificaiton ability of our new model

In [198]:
```
predictions = model.predict(X_test)
```

In [186]:
```
predictions
```

Out[186]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=uint8)
```

Metrics

In [187]:
```
from sklearn.metrics import confusion_matrix, classification_report, exp
lained_variance_score
```

In [188]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[ 4646 11078]
 [ 2003 60966]]
```

In [189]:
```
df.head(5)
```

Out[189]:

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	loan_status	dti	open_acc
0	10000.0	36	11.44	329.48	10.0	117000.0	1	26.24	16.0
1	8000.0	36	11.99	265.68	4.0	65000.0	1	22.05	17.0
2	15600.0	36	10.49	506.97	0.0	43057.0	1	12.79	13.0
3	7200.0	36	6.49	220.65	6.0	54000.0	1	2.60	6.0
4	24375.0	60	17.27	609.33	9.0	55000.0	0	33.95	13.0

5 rows × 49 columns

Model Accuracy (0: Charged off ,1: fully Paid)

- Model predicts at a 83% accuracy
- Precision or Charged off is lower since we have ar less data points for this classification, but a 70% preciaion is a good value for a prediction we had no guidence with originally

In [190]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.70      0.30      0.42     15724
           1       0.85      0.97      0.90     62969

    accuracy                           0.83     78693
   macro avg       0.77      0.63      0.66     78693
weighted avg       0.82      0.83      0.81     78693
```

Saving Model

- The performance is satisfying
- Lets store the Sklern Model using the Joblib library

In [192]:
```
import joblib
```

In [193]:
```
#path_to_save = "model/lengingTreeLogRegression.pkl"
```

In [194]:
```
#joblib.dump(model, path_to_save)
```

Out[194]:
```
['model/lengingTreeLogRegression.pkl']
```

In [191]:
```
df.corrwith(df["loan_status"]).sort_values(ascending=False)
```

Out[191]:
```
loan_status               1.000000
B                         0.114238
mort_acc                  0.069904
MORTGAGE                  0.066855
annual_inc                0.053458
credit_card               0.037622
total_acc                 0.017689
home_improvement          0.016644
issue_month               0.016287
emp_length                0.014690
wedding                   0.012821
major_purchase            0.011978
revol_bal                 0.010771
JOINT                     0.005714
earliest_cr_line_month    0.003726
OTHER                     0.002136
educational               0.002023
INDIVIDUAL                0.001800
vacation                  0.001302
house                    -0.000374
NONE                     -0.000977
renewable_energy         -0.002679
medical                  -0.005538
moving                   -0.008384
OWN                      -0.008616
pub_rec_bankruptcies     -0.009570
other                    -0.009735
initial_list_status      -0.009960
pub_rec                  -0.019984
C                        -0.024151
open_acc                 -0.028373
small_business           -0.029757
Source Verified          -0.033425
debt_consolidation       -0.034196
earliest_cr_line_year    -0.038941
installment              -0.041085
Verified                 -0.050096
loan_amnt                -0.060055
issue_year               -0.061532
dti                      -0.062186
G                        -0.062444
RENT                     -0.063045
revol_util               -0.081931
D                        -0.102108
F                        -0.102144
E                        -0.131501
term                     -0.174051
int_rate                 -0.247960
address                  -0.346973
dtype: float64
```

Appears that we have an issue with the issue_day feature

- lets go back and drop this feature
- since this was a generated feature we will keep the year and month

Deep Learning

- let try to use a deep learning model to try to improve our predicton on the target class : loan_Status

In [119]:
```
X = df.drop("loan_status", axis=1).values
y = df["loan_status"].values
```

In [121]:
```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
0, random_state=42)
```

In [122]:
```
scaler = MinMaxScaler()
```

In [123]:
```
X_train = scaler.fit_transform(X_train)
```

In [124]:
```
X_test = scaler.transform(X_test)
```

Importing Keras Libaries for model creation

- Model will be a Sequential model with approzimately 6 Hidden layers
- Dense layer will consist of units close to the count of the feature list
- Consideration for a dropout layer to assist with preventing over training
- A Early Stopping callback wil be integrate to prevent overtraining

In [125]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

In [126]:
```
stop = EarlyStopping(monitor="val_loss", mode="min", patience=15)
```

In [127]:
```
X_train.shape
```

Out[127]:
```
(314772, 48)
```

In [128]:
```
model = Sequential()
model.add(Dense(units = 50, activation = "relu"))
model.add(Dense(units = 50, activation = "relu"))
model.add(Dense(units = 50, activation = "relu"))
model.add(Dense(units = 25, activation = "relu"))
model.add(Dense(units = 25, activation = "relu"))
model.add(Dense(units = 10, activation = "relu"))
model.add(Dense(units = 10, activation = "relu"))
model.add(Dense(units = 1, activation = "sigmoid"))
model.compile(loss = "binary_crossentropy", optimizer = "adam")
```

In [129]:
```
model.fit(X_train,y_train, validation_data=(X_test, y_test), epochs=120,
callbacks=[stop])
```

```
Epoch 1/120
9837/9837 [==============================] - 17s 2ms/step - loss: 0.382
6 - val_loss: 0.2856
Epoch 2/120
9837/9837 [==============================] - 16s 2ms/step - loss: 0.282
9 - val_loss: 0.2816
Epoch 3/120
9837/9837 [==============================] - 16s 2ms/step - loss: 0.279
6 - val_loss: 0.2825
Epoch 4/120
9837/9837 [==============================] - 16s 2ms/step - loss: 0.278
4 - val_loss: 0.2832
Epoch 5/120
9837/9837 [==============================] - 16s 2ms/step - loss: 0.274
9 - val_loss: 0.2833
Epoch 6/120
9837/9837 [==============================] - 16s 2ms/step - loss: 0.273
6 - val_loss: 0.2790
Epoch 7/120
9837/9837 [==============================] - 17s 2ms/step - loss: 0.274
5 - val_loss: 0.2774
Epoch 8/120
9837/9837 [==============================] - 18s 2ms/step - loss: 0.273
8 - val_loss: 0.2765
Epoch 9/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.271
0 - val_loss: 0.2759
Epoch 10/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.271
9 - val_loss: 0.2702
Epoch 11/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.268
8 - val_loss: 0.2744
Epoch 12/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.270
5 - val_loss: 0.2803
Epoch 13/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.274
3 - val_loss: 0.2765
Epoch 14/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.269
4 - val_loss: 0.2691
Epoch 15/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.264
6 - val_loss: 0.2682
Epoch 16/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.268
4 - val_loss: 0.2675
Epoch 17/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.264
9 - val_loss: 0.2667
Epoch 18/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.263
0 - val_loss: 0.2684
Epoch 19/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.264
5 - val_loss: 0.2737
```

```
Epoch 20/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
1 - val_loss: 0.2797
Epoch 21/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.264
7 - val_loss: 0.2656
Epoch 22/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
9 - val_loss: 0.2825
Epoch 23/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.266
9 - val_loss: 0.2709
Epoch 24/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
6 - val_loss: 0.2629
Epoch 25/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.262
9 - val_loss: 0.2643
Epoch 26/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
6 - val_loss: 0.2636
Epoch 27/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.260
8 - val_loss: 0.2635
Epoch 28/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.263
1 - val_loss: 0.2624
Epoch 29/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.259
2 - val_loss: 0.2675
Epoch 30/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
1 - val_loss: 0.2615
Epoch 31/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.262
2 - val_loss: 0.2627
Epoch 32/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.260
6 - val_loss: 0.2641
Epoch 33/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.257
5 - val_loss: 0.2842
Epoch 34/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.259
7 - val_loss: 0.2802
Epoch 35/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.260
5 - val_loss: 0.2651
Epoch 36/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.258
7 - val_loss: 0.2799
Epoch 37/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.258
8 - val_loss: 0.2752
Epoch 38/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.257
7 - val_loss: 0.2753
```

```
Epoch 39/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.258
6 - val_loss: 0.2640
Epoch 40/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.260
2 - val_loss: 0.2693
Epoch 41/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.260
2 - val_loss: 0.2652
Epoch 42/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.258
1 - val_loss: 0.2644
Epoch 43/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.256
9 - val_loss: 0.2622
Epoch 44/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.257
0 - val_loss: 0.2649
Epoch 45/120
9837/9837 [==============================] - 19s 2ms/step - loss: 0.261
1 - val_loss: 0.2676
```

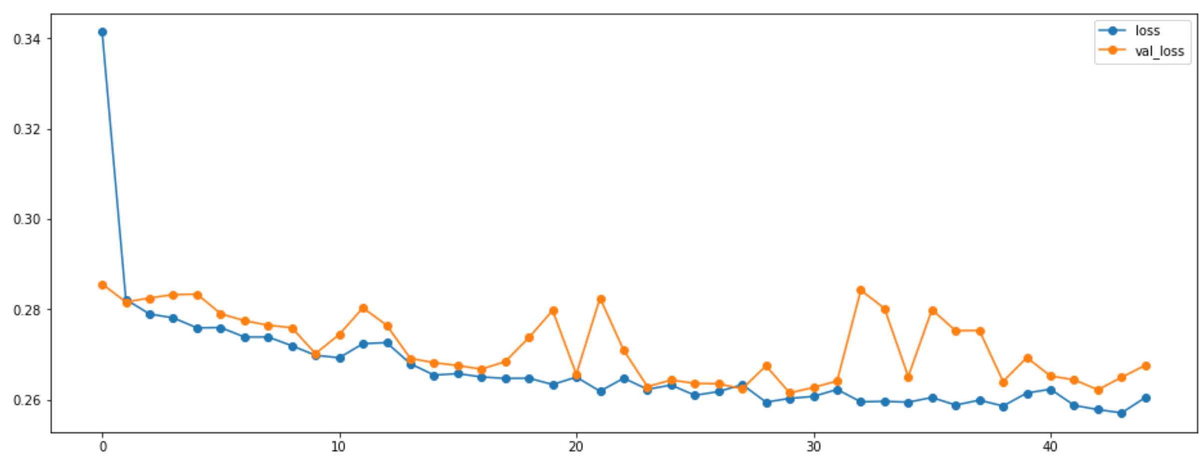
Out[129]: <tensorflow.python.keras.callbacks.History at 0x7fafb31224f0>

Model History

- Model performance was satisfying with overtraining beginning to occur around 31 epochs
- over all the training data fit well to the Validation data
- Displaying strong prediction potential for the classificaiton

In [132]:
```
pd.DataFrame(model.history.history).plot(marker = "o", figsize = (16,6))
```

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x7fafb3a7bd00>



In [134]:
```
#pd.DataFrame(model.history.history).to_csv("model/history_v1.csv")
```

Model Predicitons

- Making Predictions on the testing data using the new model
- The hope is the there is some improvment in prediciton for mthe Logistic Regression Model

In [137]:
```
predictions = (model.predict(X_test) > 0.5).astype("int32")
```

In [138]:
```
predictions
```

Out[138]:
```
array([[1],
       [1],
       [0],
       ...,
       [1],
       [1],
       [1]], dtype=int32)
```

Classification Report

- Taking a look at the performance of the new model
- It can be said that the model has a saisfying accuracy with a higher precision for Charged off but a higher recall and Fscore for Paid Off

In [139]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.47      0.62     15724
           1       0.88      0.99      0.93     62969

    accuracy                           0.89     78693
   macro avg       0.91      0.73      0.78     78693
weighted avg       0.89      0.89      0.87     78693
```

In [140]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[ 7350  8374]
 [  556 62413]]
```

Passing in Random Clients

- Since we currently do not have new clients to make prediction on, we will randomly select a client from our already established data set. Though our model was trained on this data, it is safe to say the any new clients passed will share the same features.
- This is simply another way to evaluate the models prediction capabilities

In [158]:
```
from random import randint
random_index= randint(1, len(df))
random_client = df.drop("loan_status", axis=1).iloc[random_index].values
```

Random Client

- We will need to reshape this client to the dimension our model was trained on.
- we can find this bu looking at the shape of the training data
- we will also need to scale the values to the scale of the training data

In [159]:
```
X_train.shape
```

Out[159]:
```
(314772, 48)
```

In [160]:
```
random_client = scaler.transform(random_client.reshape(1,48))
```

In [161]:
```
random_client.shape
```

Out[161]:
```
(1, 48)
```

Passing Random Client

- Passing client to the model to make a prediction

In [162]:
```
(model.predict(random_client) > 0.5).astype("int32")
```

Out[162]:
```
array([[0]], dtype=int32)
```

Checking True Value

- Getting the loan_status from the data from at selected index
- Model will predict the loan class at the above accuracy

In [163]:
```
df.iloc[random_index]["loan_status"]
```

Out[163]:
```
0.0
```

Saving model

- We are satisfied with the model performance
- Saving for future uses

In [164]:
```
##model.save("model/lendingTree_89_acc_v1.h5")
```

In [ ]:
```
```