Lending Club Loan Data Analysis

May 16, 2020

1 Importing the essential libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
```

```
Using TensorFlow backend.
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
```

```
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\harma\anaconda3\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

2 Importing the data

[2]:
```python
loan = pd.read_csv('loan_data.csv')
```

3 Exploring the data

[3]:
```python
loan.head()
```

[3]:

```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

We can observe the data contained within the dataframe

[4]:

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

As you can see all the features are numeric except for purpose

[5]:
```
loan.describe()
```

[5]:

	credit.policy	int.rate	installment	log.annual.inc	dti
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679
std	0.396245	0.026847	207.071301	0.614813	6.883970
min	0.000000	0.060000	15.670000	7.547502	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500
50%	1.000000	0.122100	268.950000	10.928884	12.665000
75%	1.000000	0.140700	432.762500	11.291293	17.950000
max	1.000000	0.216400	940.140000	14.528354	29.960000

	fico	days.with.cr.line	revol.bal	revol.util
count	9578.000000	9578.000000	9.578000e+03	9578.000000
mean	710.846314	4560.767197	1.691396e+04	46.799236
std	37.970537	2496.930377	3.375619e+04	29.014417
min	612.000000	178.958333	0.000000e+00	0.000000
25%	682.000000	2820.000000	3.187000e+03	22.600000
50%	707.000000	4139.958333	8.596000e+03	46.300000
75%	737.000000	5730.000000	1.824950e+04	70.900000
max	827.000000	17639.958330	1.207359e+06	119.000000

	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
count	9578.000000	9578.000000	9578.000000	9578.000000
mean	1.577469	0.163708	0.062122	0.160054
std	2.200245	0.546215	0.262126	0.366676
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000
75%	2.000000	0.000000	0.000000	0.000000
max	33.000000	13.000000	5.000000	1.000000

From the above computed data, we can observe many things like mean, median, standard deviation, minimum value and maximum value for all the numeric features

[6]:
```
loan.shape
```

[6]: (9578, 14)

From the above we can see the data that we imported is having 9578 rows and 14 columns

[7]:
```
loan.columns
```

[7]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

[8]:
```
loan['purpose'].value_counts()
```

[8]:
```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

[9]:
```
loan['purpose'].unique()
```

[9]:
```
array(['debt_consolidation', 'credit_card', 'all_other',
       'home_improvement', 'small_business', 'major_purchase',
       'educational'], dtype=object)
```

As you can see, we only have 7 unique value so we can convert them to numeric using label encoder

[10]:
```
le = preprocessing.LabelEncoder()
loan['purpose'] = le.fit_transform(loan['purpose'])
```

4 Exploring the changes in the data

[11]:
```
loan.head()
```

[11]:
```
   credit.policy  purpose  int.rate  installment  log.annual.inc    dti  fico  \
0              1        2    0.1189       829.10       11.350407  19.48   737
1              1        1    0.1071       228.22       11.082143  14.29   707
2              1        2    0.1357       366.86       10.373491  11.63   682
3              1        2    0.1008       162.34       11.350407   8.10   712
4              1        1    0.1426       102.92       11.299732  14.97   667

   days.with.cr.line  revol.bal  revol.util  inq.last.6mths  delinq.2yrs  \
0        5639.958333      28854        52.1               0            0
1        2760.000000      33623        76.7               0            0
2        4710.000000       3511        25.6               1            0
3        2699.958333      33667        73.2               1            0
4        4066.000000       4740        39.5               0            1

   pub.rec  not.fully.paid
0        0               0
1        0               0
2        0               0
3        0               0
```

```
4        0                0
```

We can observe that the purpose column has now been converted to numeric

[12]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   int32
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
dtypes: float64(6), int32(1), int64(7)
memory usage: 1010.3 KB
```

We can observe that the purpose is converted to int32 from object

[13]:
```
loan['purpose'].value_counts()
```

[13]:
```
2    3957
0    2331
1    1262
4     629
6     619
5     437
3     343
Name: purpose, dtype: int64
```

[14]:
```
loan['purpose'].unique()
```

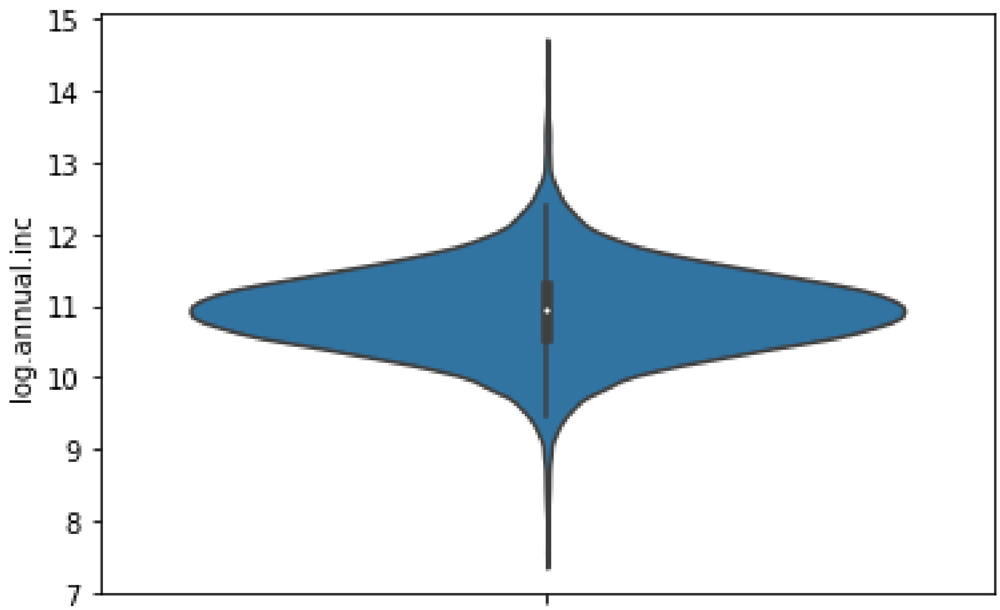
[14]:
```
array([2, 1, 0, 4, 6, 5, 3])
```

We can observe that after label encoding the purpose column, we have now converted string data to numeric

5 Exploring the data visually

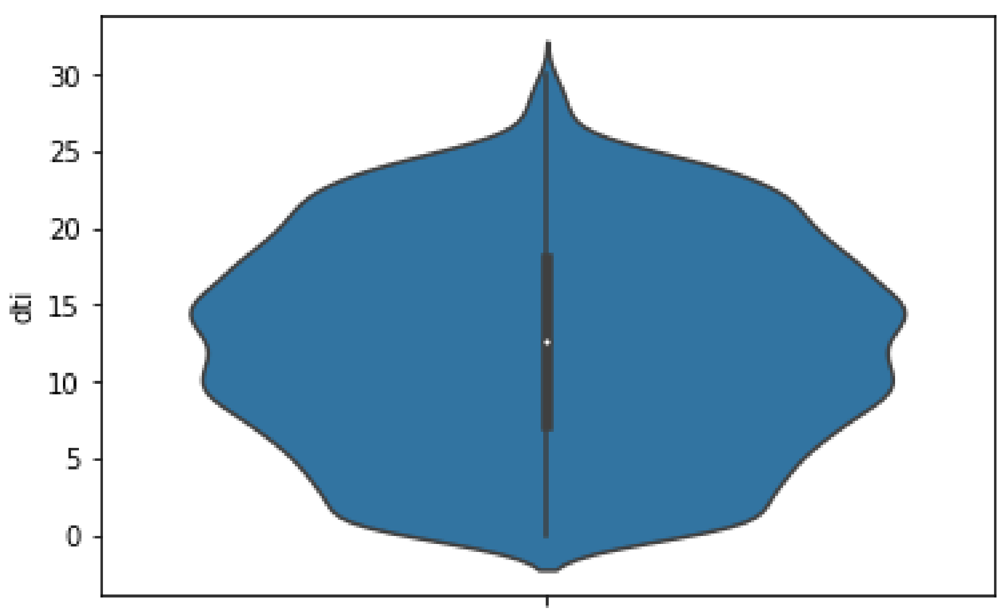
[15]:
```
sns.violinplot(y = 'log.annual.inc', hue='not.fully.paid',data = loan)
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x250b7103b08>



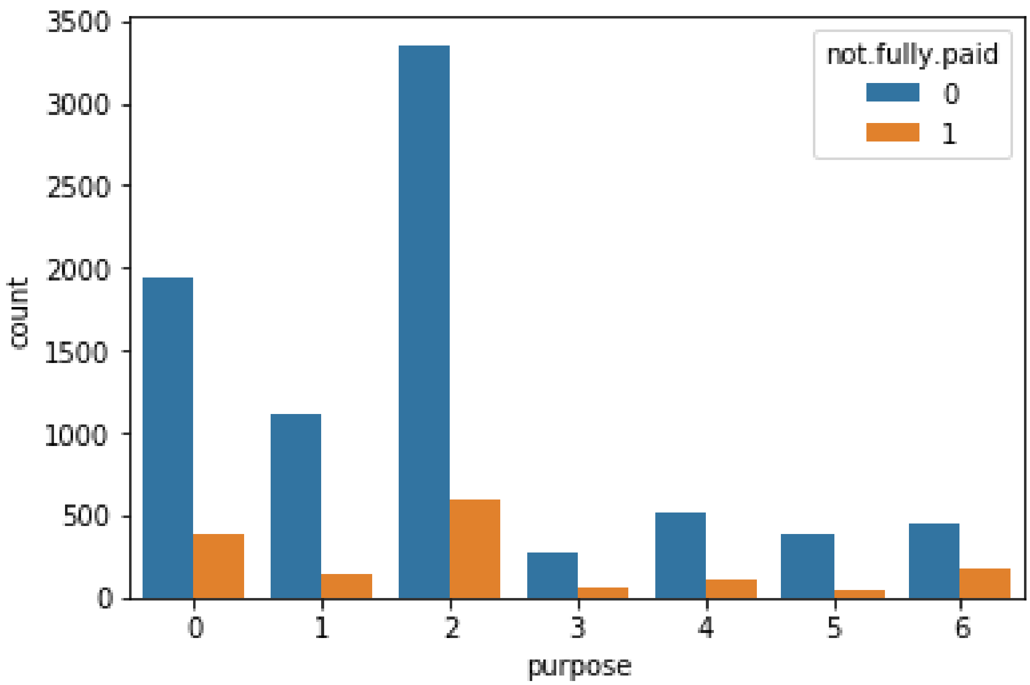
[16]:
```
sns.violinplot(y='dti',hue='not.fully.paid',data = loan)
```

[16]: <matplotlib.axes._subplots.AxesSubplot at 0x250b78b7948>



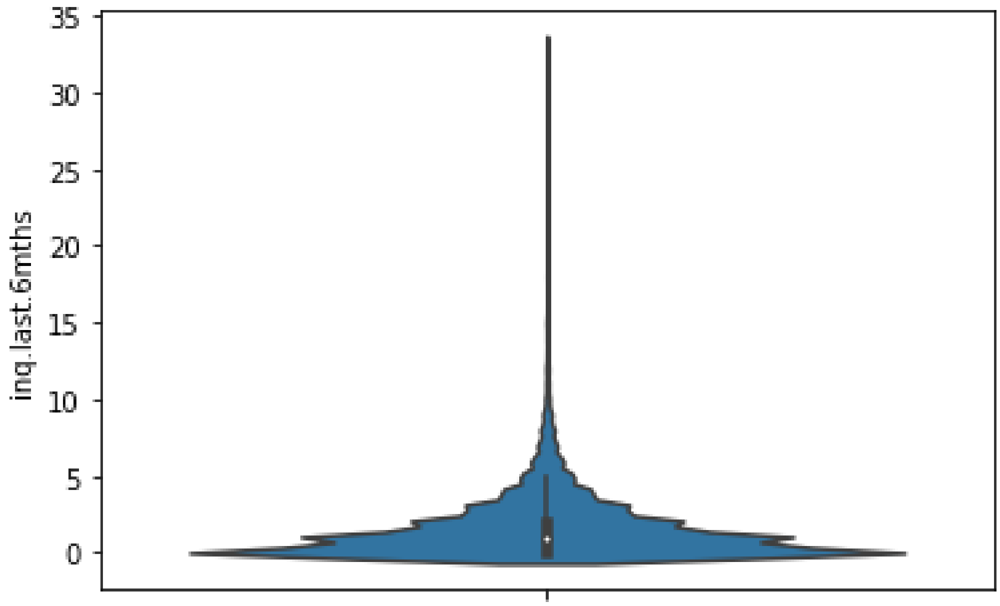
[17]:
```
sns.countplot(x='purpose',hue='not.fully.paid',data = loan)
```

[17]: <matplotlib.axes._subplots.AxesSubplot at 0x250b791ff48>



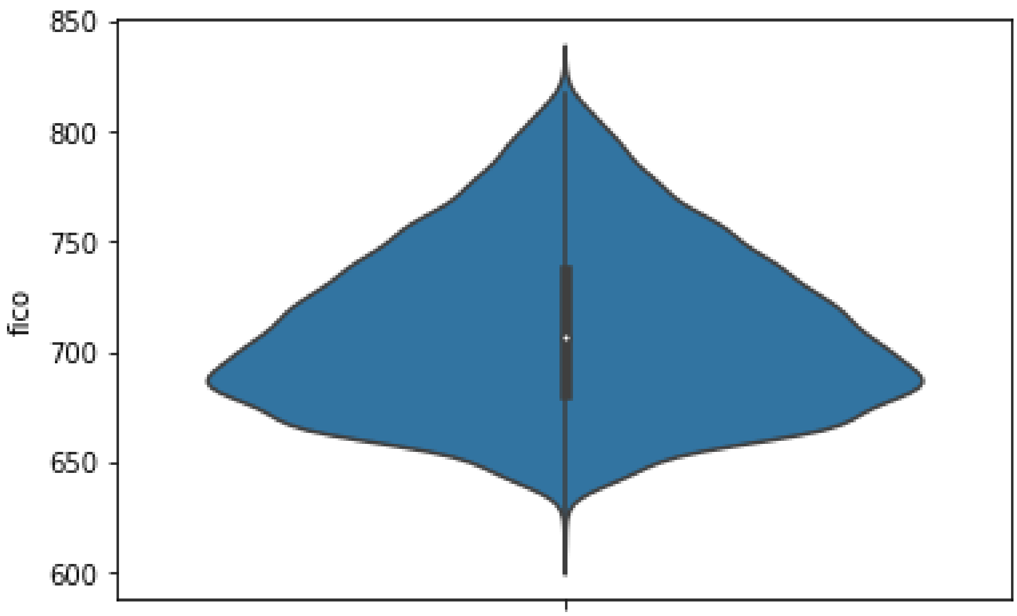
[18]:
```
sns.violinplot(y='inq.last.6mths',hue='not.fully.paid',data = loan)
```

[18]: <matplotlib.axes._subplots.AxesSubplot at 0x250b79d4ac8>



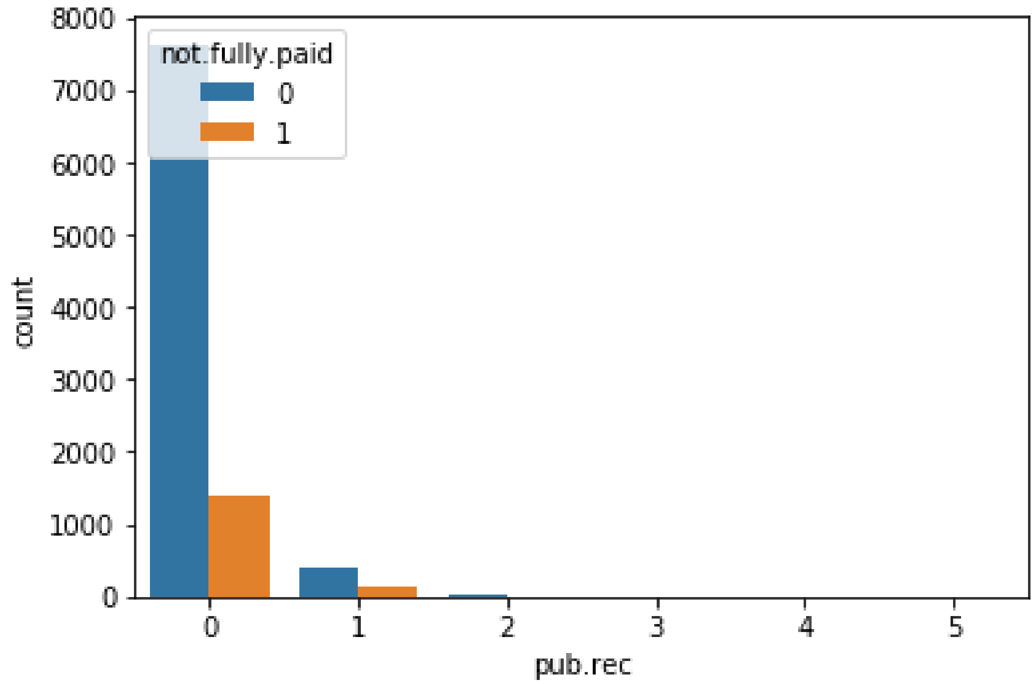
[19]:
```
sns.violinplot(y='fico',hue='not.fully.paid',data = loan)
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x250b7a49d08>



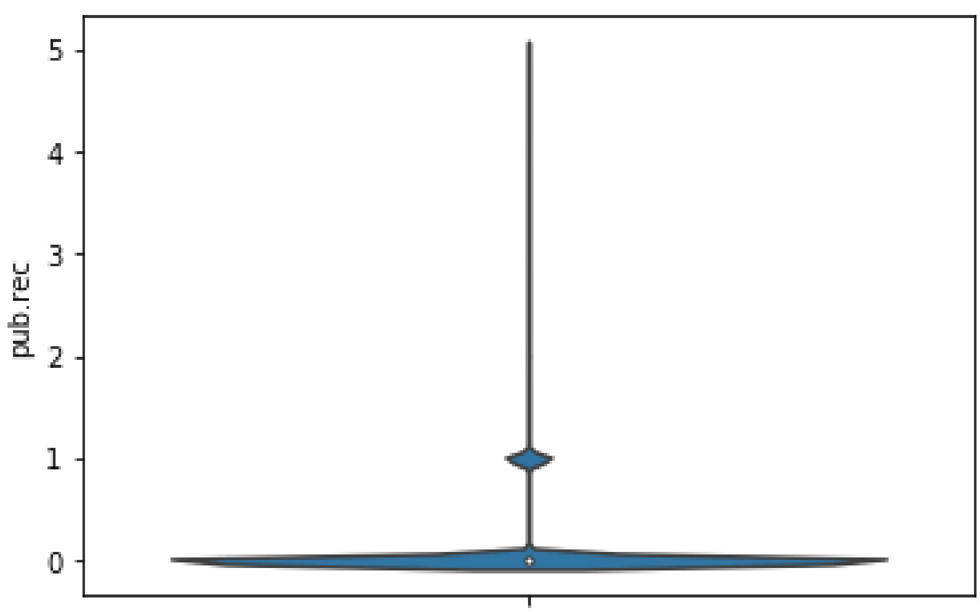
[20]:
```
sns.countplot(x='pub.rec',hue='not.fully.paid',data = loan)
```

```
[20]: <matplotlib.axes._subplots.AxesSubplot at 0x250b7a98ec8>
```



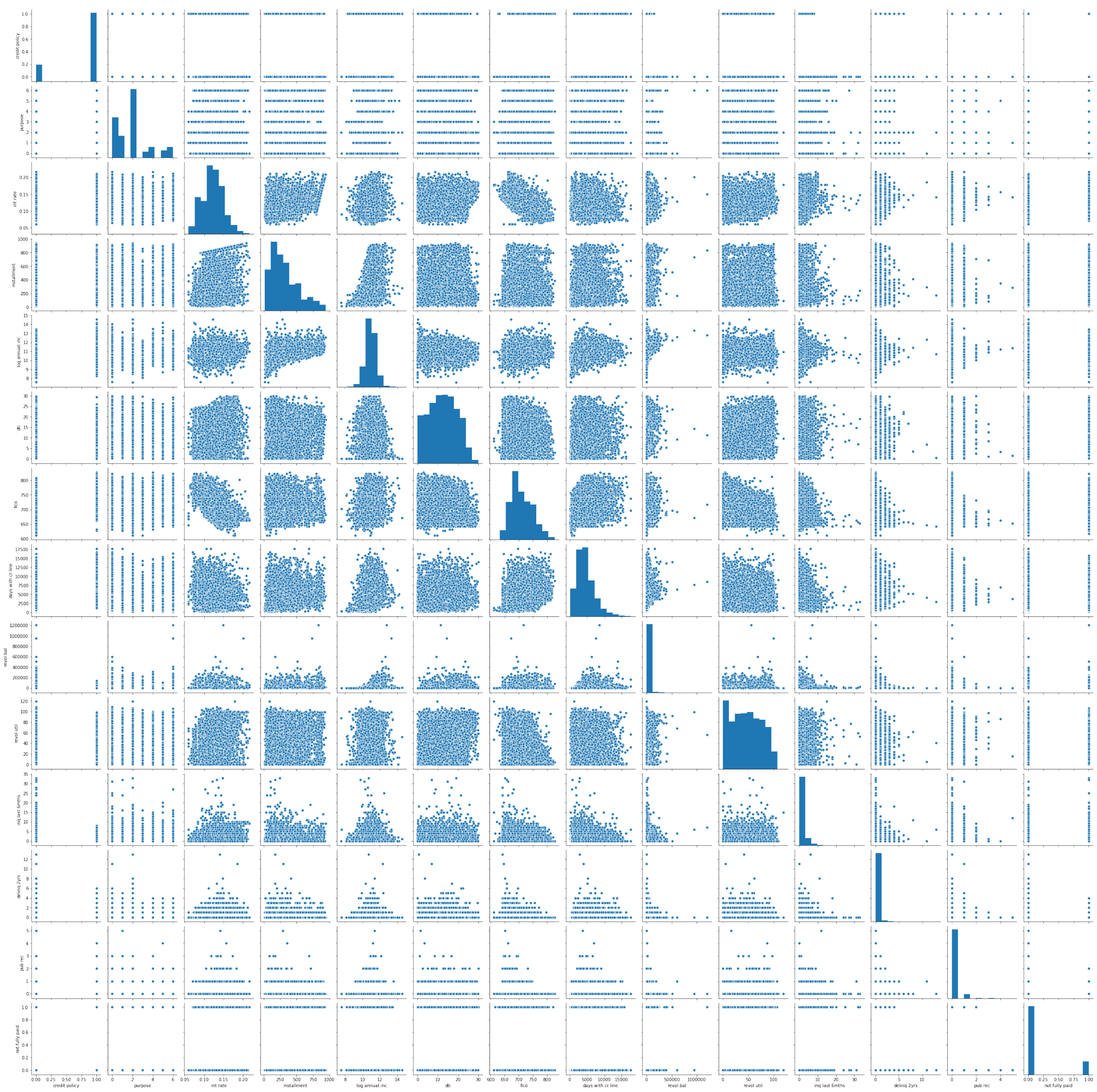

```
[21]: sns.violinplot(y='pub.rec',hue='not.fully.paid',data = loan)
```

```
[21]: <matplotlib.axes._subplots.AxesSubplot at 0x250b7b28508>
```



[23]:
```
sns.pairplot(data = loan)
```

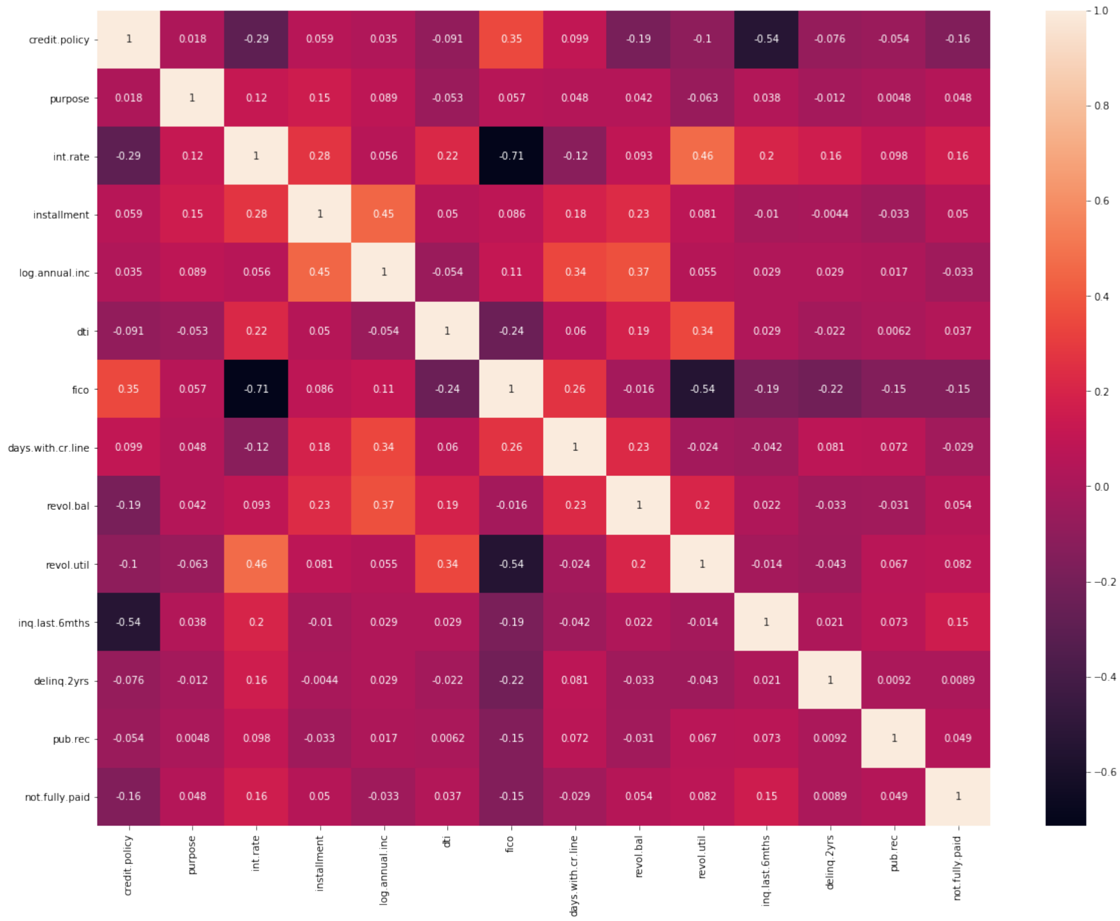
[23]: <seaborn.axisgrid.PairGrid at 0x250bf86d688>



6 Checking the correlations between features

[24]:
```
corrMatrix = loan.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corrMatrix, annot=True)
```

[24]: <matplotlib.axes._subplots.AxesSubplot at 0x250cadcd448>



We can see many significant correlations from above heatmap

7 Loading inputs and labels

[25]:
```
input = pd.read_csv('input.csv')
input.shape
```

[25]: (9577, 18)

[26]:
```
labels = pd.read_csv('output.csv')
labels.shape
```

[26]: (9577, 2)

8 Splitting the data into training and testing data

[27]:
```
x_train, x_test, y_train, y_test = train_test_split(input,labels,test_size = 0.
↪25)
```

[28]:
```
x_train.shape
```

[28]: (7182, 18)

[29]:
```
y_train.shape
```

[29]: (7182, 2)

[30]:
```
x_test.shape
```

[30]: (2395, 18)

[31]:
```
y_test.shape
```

[31]: (2395, 2)

9 Creating the model

[32]:
```
model = Sequential()

model.add(Dense(12,input_dim=18,activation='relu'))
model.add(Dense(18, activation='relu'))
model.add(Dense(2,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(x_train,y_train,epochs=200,batch_size=10)

scores = model.evaluate(x_test,y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
WARNING:tensorflow:From C:\Users\harma\anaconda3\lib\site-
packages\tensorflow\python\ops\nn_impl.py:180:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\harma\anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.
```

```
Epoch 1/200
7182/7182 [==============================] - 1s 144us/step - loss: 61.7472 -
accuracy: 0.8042
Epoch 2/200
7182/7182 [==============================] - 1s 103us/step - loss: 13.4713 -
accuracy: 0.8608
Epoch 3/200
7182/7182 [==============================] - 1s 108us/step - loss: 16.0615 -
accuracy: 0.8743
Epoch 4/200
7182/7182 [==============================] - 1s 103us/step - loss: 14.0060 -
accuracy: 0.8761
Epoch 5/200
7182/7182 [==============================] - 1s 104us/step - loss: 8.5710 -
accuracy: 0.8796
Epoch 6/200
7182/7182 [==============================] - 1s 92us/step - loss: 11.6795 -
accuracy: 0.8768
Epoch 7/200
7182/7182 [==============================] - 1s 101us/step - loss: 8.4640 -
accuracy: 0.8813
Epoch 8/200
7182/7182 [==============================] - 1s 156us/step - loss: 7.7313 -
accuracy: 0.8800
Epoch 9/200
7182/7182 [==============================] - 1s 160us/step - loss: 8.6938 -
accuracy: 0.8810
Epoch 10/200
7182/7182 [==============================] - 1s 132us/step - loss: 10.3051 -
accuracy: 0.8749
Epoch 11/200
7182/7182 [==============================] - 1s 131us/step - loss: 11.0316 -
accuracy: 0.8749
Epoch 12/200
7182/7182 [==============================] - 1s 123us/step - loss: 7.9946 -
accuracy: 0.8794
Epoch 13/200
7182/7182 [==============================] - 1s 122us/step - loss: 7.0065 -
accuracy: 0.8807
Epoch 14/200
7182/7182 [==============================] - 1s 106us/step - loss: 5.8363 -
accuracy: 0.8813
Epoch 15/200
7182/7182 [==============================] - 1s 105us/step - loss: 5.5541 -
accuracy: 0.8803
Epoch 16/200
7182/7182 [==============================] - 1s 106us/step - loss: 6.3712 -
accuracy: 0.8786
```

```
Epoch 17/200
7182/7182 [==============================] - 1s 108us/step - loss: 6.0628 -
accuracy: 0.8801
Epoch 18/200
7182/7182 [==============================] - 1s 110us/step - loss: 4.6333 -
accuracy: 0.8823
Epoch 19/200
7182/7182 [==============================] - 1s 101us/step - loss: 4.1560 -
accuracy: 0.8829
Epoch 20/200
7182/7182 [==============================] - 1s 98us/step - loss: 4.4054 -
accuracy: 0.8816
Epoch 21/200
7182/7182 [==============================] - 1s 98us/step - loss: 4.3465 -
accuracy: 0.8789
Epoch 22/200
7182/7182 [==============================] - 1s 95us/step - loss: 3.2615 -
accuracy: 0.8833
Epoch 23/200
7182/7182 [==============================] - 1s 104us/step - loss: 2.7786 -
accuracy: 0.8760
Epoch 24/200
7182/7182 [==============================] - 1s 97us/step - loss: 3.0418 -
accuracy: 0.8796
Epoch 25/200
7182/7182 [==============================] - 1s 94us/step - loss: 2.4707 -
accuracy: 0.8826
Epoch 26/200
7182/7182 [==============================] - 1s 100us/step - loss: 2.2122 -
accuracy: 0.8866
Epoch 27/200
7182/7182 [==============================] - 1s 92us/step - loss: 2.5340 -
accuracy: 0.8857
Epoch 28/200
7182/7182 [==============================] - 1s 116us/step - loss: 1.7052 -
accuracy: 0.8752
Epoch 29/200
7182/7182 [==============================] - 1s 149us/step - loss: 2.3936 -
accuracy: 0.8810
Epoch 30/200
7182/7182 [==============================] - 1s 124us/step - loss: 2.3151 -
accuracy: 0.8865
Epoch 31/200
7182/7182 [==============================] - 1s 124us/step - loss: 1.6869 -
accuracy: 0.8938
Epoch 32/200
7182/7182 [==============================] - 1s 124us/step - loss: 1.5806 -
accuracy: 0.8878
```

```
Epoch 33/200
7182/7182 [==============================] - 1s 115us/step - loss: 1.5058 -
accuracy: 0.8936
Epoch 34/200
7182/7182 [==============================] - 1s 119us/step - loss: 1.5755 -
accuracy: 0.8967
Epoch 35/200
7182/7182 [==============================] - 1s 117us/step - loss: 1.4627 -
accuracy: 0.8991
Epoch 36/200
7182/7182 [==============================] - 1s 116us/step - loss: 1.5422 -
accuracy: 0.9031
Epoch 37/200
7182/7182 [==============================] - 1s 117us/step - loss: 1.5530 -
accuracy: 0.8943
Epoch 38/200
7182/7182 [==============================] - 1s 101us/step - loss: 1.4964 -
accuracy: 0.9062
Epoch 39/200
7182/7182 [==============================] - 1s 89us/step - loss: 1.4505 -
accuracy: 0.9046
Epoch 40/200
7182/7182 [==============================] - 1s 90us/step - loss: 1.0022 -
accuracy: 0.9087
Epoch 41/200
7182/7182 [==============================] - 1s 115us/step - loss: 0.9394 -
accuracy: 0.9078
Epoch 42/200
7182/7182 [==============================] - 1s 120us/step - loss: 1.0177 -
accuracy: 0.9087
Epoch 43/200
7182/7182 [==============================] - 1s 117us/step - loss: 1.0725 -
accuracy: 0.9101
Epoch 44/200
7182/7182 [==============================] - 1s 133us/step - loss: 0.7842 -
accuracy: 0.9080
Epoch 45/200
7182/7182 [==============================] - 1s 121us/step - loss: 0.7502 -
accuracy: 0.9165
Epoch 46/200
7182/7182 [==============================] - 1s 125us/step - loss: 0.6380 -
accuracy: 0.9192
Epoch 47/200
7182/7182 [==============================] - 1s 121us/step - loss: 0.5479 -
accuracy: 0.9231
Epoch 48/200
7182/7182 [==============================] - 1s 121us/step - loss: 0.4486 -
accuracy: 0.9229
```

```
Epoch 49/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.3858 -
accuracy: 0.9260
Epoch 50/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.3369 -
accuracy: 0.9282
Epoch 51/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2810 -
accuracy: 0.9311
Epoch 52/200
7182/7182 [==============================] - 1s 88us/step - loss: 0.2621 -
accuracy: 0.9317
Epoch 53/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2481 -
accuracy: 0.9319
Epoch 54/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2475 -
accuracy: 0.9321
Epoch 55/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2484 -
accuracy: 0.9321
Epoch 56/200
7182/7182 [==============================] - 1s 90us/step - loss: 0.2479 -
accuracy: 0.9321
Epoch 57/200
7182/7182 [==============================] - 1s 86us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 58/200
7182/7182 [==============================] - 1s 85us/step - loss: 0.2494 -
accuracy: 0.9321
Epoch 59/200
7182/7182 [==============================] - 1s 85us/step - loss: 0.2482 -
accuracy: 0.9321
Epoch 60/200
7182/7182 [==============================] - 1s 88us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 61/200
7182/7182 [==============================] - 1s 88us/step - loss: 0.2479 -
accuracy: 0.9321
Epoch 62/200
7182/7182 [==============================] - 1s 91us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 63/200
7182/7182 [==============================] - 1s 91us/step - loss: 0.2607 -
accuracy: 0.9320
Epoch 64/200
7182/7182 [==============================] - 1s 86us/step - loss: 0.2485 -
accuracy: 0.9321
```

```
Epoch 65/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2483 -
accuracy: 0.9321
Epoch 66/200
7182/7182 [==============================] - 1s 111us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 67/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 68/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 69/200
7182/7182 [==============================] - 1s 121us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 70/200
7182/7182 [==============================] - 1s 113us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 71/200
7182/7182 [==============================] - 1s 113us/step - loss: 0.2482 -
accuracy: 0.9321
Epoch 72/200
7182/7182 [==============================] - 1s 101us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 73/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 74/200
7182/7182 [==============================] - 1s 125us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 75/200
7182/7182 [==============================] - 1s 135us/step - loss: 0.2482 -
accuracy: 0.9321
Epoch 76/200
7182/7182 [==============================] - 1s 110us/step - loss: 0.2479 -
accuracy: 0.9321
Epoch 77/200
7182/7182 [==============================] - 1s 110us/step - loss: 0.2481 -
accuracy: 0.9321
Epoch 78/200
7182/7182 [==============================] - 1s 122us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 79/200
7182/7182 [==============================] - 1s 115us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 80/200
7182/7182 [==============================] - 1s 114us/step - loss: 0.2480 -
accuracy: 0.9321
```

```
Epoch 81/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 82/200
7182/7182 [==============================] - 1s 123us/step - loss: 0.2475 -
accuracy: 0.9321
Epoch 83/200
7182/7182 [==============================] - 1s 118us/step - loss: 0.2486 -
accuracy: 0.9321
Epoch 84/200
7182/7182 [==============================] - 1s 114us/step - loss: 0.2485 -
accuracy: 0.9318
Epoch 85/200
7182/7182 [==============================] - 1s 107us/step - loss: 0.2475 -
accuracy: 0.9321
Epoch 86/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 87/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2476 -
accuracy: 0.9321
Epoch 88/200
7182/7182 [==============================] - 1s 181us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 89/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2476 -
accuracy: 0.9321
Epoch 90/200
7182/7182 [==============================] - 1s 147us/step - loss: 0.2474 -
accuracy: 0.9321
Epoch 91/200
7182/7182 [==============================] - 1s 190us/step - loss: 0.2484 -
accuracy: 0.9320
Epoch 92/200
7182/7182 [==============================] - 1s 131us/step - loss: 0.2477 -
accuracy: 0.9321
Epoch 93/200
7182/7182 [==============================] - 1s 122us/step - loss: 0.2483 -
accuracy: 0.9321
Epoch 94/200
7182/7182 [==============================] - 1s 127us/step - loss: 0.2474 -
accuracy: 0.93210s -
Epoch 95/200
7182/7182 [==============================] - 1s 119us/step - loss: 0.2472 -
accuracy: 0.9321
Epoch 96/200
7182/7182 [==============================] - 1s 109us/step - loss: 0.2453 -
accuracy: 0.9321
```

```
Epoch 97/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2463 -
accuracy: 0.9321
Epoch 98/200
7182/7182 [==============================] - 1s 139us/step - loss: 0.2472 -
accuracy: 0.9321
Epoch 99/200
7182/7182 [==============================] - 1s 143us/step - loss: 0.2464 -
accuracy: 0.9321
Epoch 100/200
7182/7182 [==============================] - 1s 122us/step - loss: 0.2463 -
accuracy: 0.9319
Epoch 101/200
7182/7182 [==============================] - 1s 140us/step - loss: 0.2469 -
accuracy: 0.9321
Epoch 102/200
7182/7182 [==============================] - 1s 129us/step - loss: 0.2484 -
accuracy: 0.9321
Epoch 103/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2467 -
accuracy: 0.9321
Epoch 104/200
7182/7182 [==============================] - 1s 110us/step - loss: 0.2465 -
accuracy: 0.9321
Epoch 105/200
7182/7182 [==============================] - 1s 104us/step - loss: 0.2470 -
accuracy: 0.9321
Epoch 106/200
7182/7182 [==============================] - 1s 103us/step - loss: 0.2467 -
accuracy: 0.9321
Epoch 107/200
7182/7182 [==============================] - 1s 104us/step - loss: 0.2452 -
accuracy: 0.9321
Epoch 108/200
7182/7182 [==============================] - 1s 104us/step - loss: 0.2447 -
accuracy: 0.9321
Epoch 109/200
7182/7182 [==============================] - 1s 101us/step - loss: 0.2457 -
accuracy: 0.9321
Epoch 110/200
7182/7182 [==============================] - 1s 97us/step - loss: 0.2428 -
accuracy: 0.9321
Epoch 111/200
7182/7182 [==============================] - 1s 91us/step - loss: 0.2425 -
accuracy: 0.9321
Epoch 112/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2430 -
accuracy: 0.9321
```

```
Epoch 113/200
7182/7182 [==============================] - 1s 99us/step - loss: 0.2430 -
accuracy: 0.9321
Epoch 114/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2442 -
accuracy: 0.9321
Epoch 115/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2429 -
accuracy: 0.9321
Epoch 116/200
7182/7182 [==============================] - 1s 100us/step - loss: 0.2419 -
accuracy: 0.9321
Epoch 117/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2416 -
accuracy: 0.9321
Epoch 118/200
7182/7182 [==============================] - 1s 98us/step - loss: 0.2385 -
accuracy: 0.9321
Epoch 119/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2408 -
accuracy: 0.9321
Epoch 120/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2389 -
accuracy: 0.9321
Epoch 121/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2388 -
accuracy: 0.9321
Epoch 122/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2466 -
accuracy: 0.9321
Epoch 123/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2400 -
accuracy: 0.9321
Epoch 124/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2417 -
accuracy: 0.9321
Epoch 125/200
7182/7182 [==============================] - 1s 100us/step - loss: 0.2413 -
accuracy: 0.9321
Epoch 126/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2420 -
accuracy: 0.9321
Epoch 127/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2400 -
accuracy: 0.9321
Epoch 128/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2402 -
accuracy: 0.9321
```

```
Epoch 129/200
7182/7182 [==============================] - 1s 105us/step - loss: 0.2470 -
accuracy: 0.9321
Epoch 130/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2453 -
accuracy: 0.9321
Epoch 131/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2431 -
accuracy: 0.9321
Epoch 132/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2417 -
accuracy: 0.9321
Epoch 133/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2412 -
accuracy: 0.9321
Epoch 134/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2430 -
accuracy: 0.9321
Epoch 135/200
7182/7182 [==============================] - 1s 101us/step - loss: 0.2427 -
accuracy: 0.9321
Epoch 136/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2417 -
accuracy: 0.9321
Epoch 137/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2394 -
accuracy: 0.9321
Epoch 138/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2385 -
accuracy: 0.9321
Epoch 139/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2412 -
accuracy: 0.9321
Epoch 140/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2420 -
accuracy: 0.9321
Epoch 141/200
7182/7182 [==============================] - 1s 98us/step - loss: 0.2455 -
accuracy: 0.9321
Epoch 142/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2448 -
accuracy: 0.9321
Epoch 143/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2395 -
accuracy: 0.9321
Epoch 144/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2433 -
accuracy: 0.9321
```

```
Epoch 145/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2434 -
accuracy: 0.9321
Epoch 146/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2445 -
accuracy: 0.9321
Epoch 147/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2441 -
accuracy: 0.9321
Epoch 148/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2434 -
accuracy: 0.9321
Epoch 149/200
7182/7182 [==============================] - 1s 101us/step - loss: 0.2416 -
accuracy: 0.9321
Epoch 150/200
7182/7182 [==============================] - 1s 97us/step - loss: 0.2462 -
accuracy: 0.9321
Epoch 151/200
7182/7182 [==============================] - 1s 102us/step - loss: 0.2452 -
accuracy: 0.9321
Epoch 152/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2449 -
accuracy: 0.9321
Epoch 153/200
7182/7182 [==============================] - 1s 96us/step - loss: 0.2439 -
accuracy: 0.9321
Epoch 154/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2401 -
accuracy: 0.9321
Epoch 155/200
7182/7182 [==============================] - 1s 126us/step - loss: 0.2424 -
accuracy: 0.9321
Epoch 156/200
7182/7182 [==============================] - 1s 137us/step - loss: 0.2394 -
accuracy: 0.9321
Epoch 157/200
7182/7182 [==============================] - 1s 142us/step - loss: 0.2404 -
accuracy: 0.9321
Epoch 158/200
7182/7182 [==============================] - 1s 119us/step - loss: 0.2438 -
accuracy: 0.9321
Epoch 159/200
7182/7182 [==============================] - 1s 127us/step - loss: 0.2464 -
accuracy: 0.9321
Epoch 160/200
7182/7182 [==============================] - 1s 125us/step - loss: 0.2416 -
accuracy: 0.9321
```

```
Epoch 161/200
7182/7182 [==============================] - 1s 132us/step - loss: 0.2456 -
accuracy: 0.9321
Epoch 162/200
7182/7182 [==============================] - 1s 137us/step - loss: 0.2445 -
accuracy: 0.9321
Epoch 163/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2388 -
accuracy: 0.9321
Epoch 164/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2422 -
accuracy: 0.9320
Epoch 165/200
7182/7182 [==============================] - 1s 118us/step - loss: 0.2407 -
accuracy: 0.9321
Epoch 166/200
7182/7182 [==============================] - 1s 101us/step - loss: 0.2395 -
accuracy: 0.9321
Epoch 167/200
7182/7182 [==============================] - 1s 94us/step - loss: 0.2438 -
accuracy: 0.9321
Epoch 168/200
7182/7182 [==============================] - 1s 90us/step - loss: 0.2425 -
accuracy: 0.9321
Epoch 169/200
7182/7182 [==============================] - 1s 89us/step - loss: 0.2416 -
accuracy: 0.9321
Epoch 170/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2383 -
accuracy: 0.9321
Epoch 171/200
7182/7182 [==============================] - 1s 88us/step - loss: 0.2480 -
accuracy: 0.9321
Epoch 172/200
7182/7182 [==============================] - 1s 128us/step - loss: 0.2466 -
accuracy: 0.9321
Epoch 173/200
7182/7182 [==============================] - 1s 122us/step - loss: 0.2478 -
accuracy: 0.9321
Epoch 174/200
7182/7182 [==============================] - 1s 120us/step - loss: 0.2455 -
accuracy: 0.9321
Epoch 175/200
7182/7182 [==============================] - 1s 123us/step - loss: 0.2448 -
accuracy: 0.9321
Epoch 176/200
7182/7182 [==============================] - 1s 118us/step - loss: 0.2411 -
accuracy: 0.9321
```

```
Epoch 177/200
7182/7182 [==============================] - 1s 141us/step - loss: 0.2363 -
accuracy: 0.9321
Epoch 178/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2381 -
accuracy: 0.9321
Epoch 179/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2411 -
accuracy: 0.9321
Epoch 180/200
7182/7182 [==============================] - 1s 124us/step - loss: 0.2368 -
accuracy: 0.9321
Epoch 181/200
7182/7182 [==============================] - 1s 119us/step - loss: 0.2371 -
accuracy: 0.9321
Epoch 182/200
7182/7182 [==============================] - 1s 129us/step - loss: 0.2386 -
accuracy: 0.9321
Epoch 183/200
7182/7182 [==============================] - 1s 126us/step - loss: 0.2401 -
accuracy: 0.9321
Epoch 184/200
7182/7182 [==============================] - 1s 144us/step - loss: 0.2389 -
accuracy: 0.9321
Epoch 185/200
7182/7182 [==============================] - 1s 140us/step - loss: 0.2442 -
accuracy: 0.9321
Epoch 186/200
7182/7182 [==============================] - 1s 132us/step - loss: 0.2378 -
accuracy: 0.9321
Epoch 187/200
7182/7182 [==============================] - 1s 91us/step - loss: 0.2363 -
accuracy: 0.9321
Epoch 188/200
7182/7182 [==============================] - 1s 114us/step - loss: 0.2394 -
accuracy: 0.9321
Epoch 189/200
7182/7182 [==============================] - 1s 132us/step - loss: 0.2429 -
accuracy: 0.9321
Epoch 190/200
7182/7182 [==============================] - 1s 140us/step - loss: 0.2411 -
accuracy: 0.9321
Epoch 191/200
7182/7182 [==============================] - 1s 126us/step - loss: 0.2373 -
accuracy: 0.9321
Epoch 192/200
7182/7182 [==============================] - 1s 136us/step - loss: 0.2412 -
accuracy: 0.9321
```

```
Epoch 193/200
7182/7182 [==============================] - 1s 137us/step - loss: 0.2405 -
accuracy: 0.9321
Epoch 194/200
7182/7182 [==============================] - 1s 127us/step - loss: 0.2407 -
accuracy: 0.9321
Epoch 195/200
7182/7182 [==============================] - 1s 129us/step - loss: 0.2375 -
accuracy: 0.9321
Epoch 196/200
7182/7182 [==============================] - 1s 117us/step - loss: 0.2417 -
accuracy: 0.9321
Epoch 197/200
7182/7182 [==============================] - 1s 88us/step - loss: 0.2381 -
accuracy: 0.9321
Epoch 198/200
7182/7182 [==============================] - 1s 93us/step - loss: 0.2383 -
accuracy: 0.9321
Epoch 199/200
7182/7182 [==============================] - 1s 92us/step - loss: 0.2398 -
accuracy: 0.9321
Epoch 200/200
7182/7182 [==============================] - 1s 95us/step - loss: 0.2361 -
accuracy: 0.9321
2395/2395 [==============================] - 0s 29us/step

accuracy: 94.53%
```

[36]:

```python
predictions = model.predict(x_test)
y_rounded = [round(x[0]) for x in predictions]
scores_test = model.evaluate(x_test,y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores_test[1]*100))
```

```
2395/2395 [==============================] - 0s 14us/step

accuracy: 94.53%
```

We are running the above model for 200 iterations(epochs) and getting a accuracy of 94.53% on testing data