Projeto árvores de decisão e florestas aleatórias

October 8, 2024

1 Projeto florestas aleatórias

Para este projeto, estaremos explorando dados disponíveis publicamente de LendingClub.com. Lending Club conecta pessoas que precisam de dinheiro (mutuários) com pessoas que têm dinheiro (investidores). Felizmente, como investidor, você gostaria de investir em pessoas que mostraram um perfil de ter uma alta probabilidade de pagá-lo de volta. Vamos tentar criar um modelo que ajude a prever isso.

O clube de empréstimo teve um ano muito interessante em 2016, então vamos verificar alguns de seus dados e ter em mente o contexto. Esses dados são de antes mesmo de serem públicos.

Utilizaremos os dados de empréstimos de 2007-2010 e tentaremos classificar e prever se o mutuário pagou o empréstimo na íntegra. Você pode baixar os dados de aqui ou apenas usar o csv já fornecido. Recomenda-se que você use o csv fornecido, uma vez que foi limpo dos valores de NA.

Aqui estão o que as colunas representam: * credit.policy: 1 se o cliente atender aos critérios de subscrição de crédito da LendingClub.com e 0 caso contrário. * purpose: O objetivo do empréstimo (leva valores "credit_card", "debt_consolidation", "educacional", "grande compra", "small_business" e "all_other"). * int.rate: a taxa de juros do empréstimo (uma taxa de 11% seria armazenada como 0,11). Os mutuários julgados por LendingClub.com para serem mais arriscados recebem taxas de juros mais elevadas. * installment: as parcelas mensais devidas pelo mutuário se o empréstimo for financiado. * log.annual.inc: O log natural da renda anual auto-relatada do mutuário. * dti: Ratio dívida / rendimento do tomador do empréstimo (montante da dívida dividido pela receita anual). * fico: a pontuação de crédito FICO do mutuário. * days.with.cr.line: O número de dias em que o mutuário teve uma linha de crédito. * revol.bal: Saldo rotativo do mutuário (montante não pago no final do ciclo de cobrança do cartão de crédito). * revol.util: taxa de utilização da linha rotativa do mutuário (o valor da linha de crédito usada em relação ao crédito total disponível). * inq.last.6mths: número de consultas do mutuário por credores nos últimos 6 meses. * delinq.2yrs: o número de vezes que o mutuário havia passado mais de 30 dias em um pagamento nos últimos 2 anos. * pub.rec: O número de registros públicos depreciativos do mutuário (arquivamentos de falências, ônus fiscais ou julgamentos).

2 Importar bibliotecas

** Importe as bibliotecas usuais para pandas e plotagem. Você pode importar sklearn mais tarde. **

[2]:
```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

2.1 Obter dados

** Use pandas para ler loan_data.csv como um DataFrame chamado loans. **

[3]:
```
loans = pd.read_csv('loan_data.csv')
```

** Use os métodos info(), head(), e describe() em loans. **

[4]:
```
loans.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
```

[5]:
```
loans.head()
```

[5]:

	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
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

```
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
```

[6]:
```
loans.describe()
```

[6]:
```
       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
```

3 Análise exploratória de dados

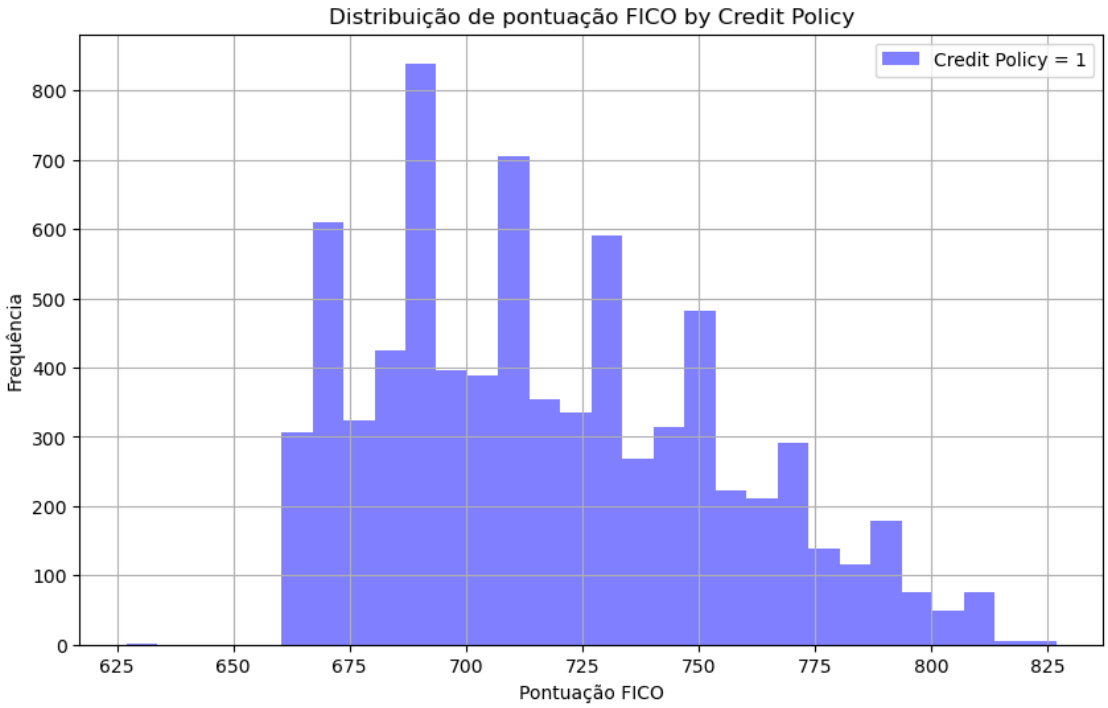
Vamos fazer alguma visualização de dados! Usaremos os recursos de plotagem incorporados ao seaborn e ao pandas, mas sinta-se livre para usar qualquer biblioteca que você deseja. Não se preocupe com as cores, apenas se preocupe em obter a idéia principal do plot.

** Crie um histograma de duas distribuições FICO umas sobre as outras, uma para cada um dos

valores possíveis de credit.policy **.

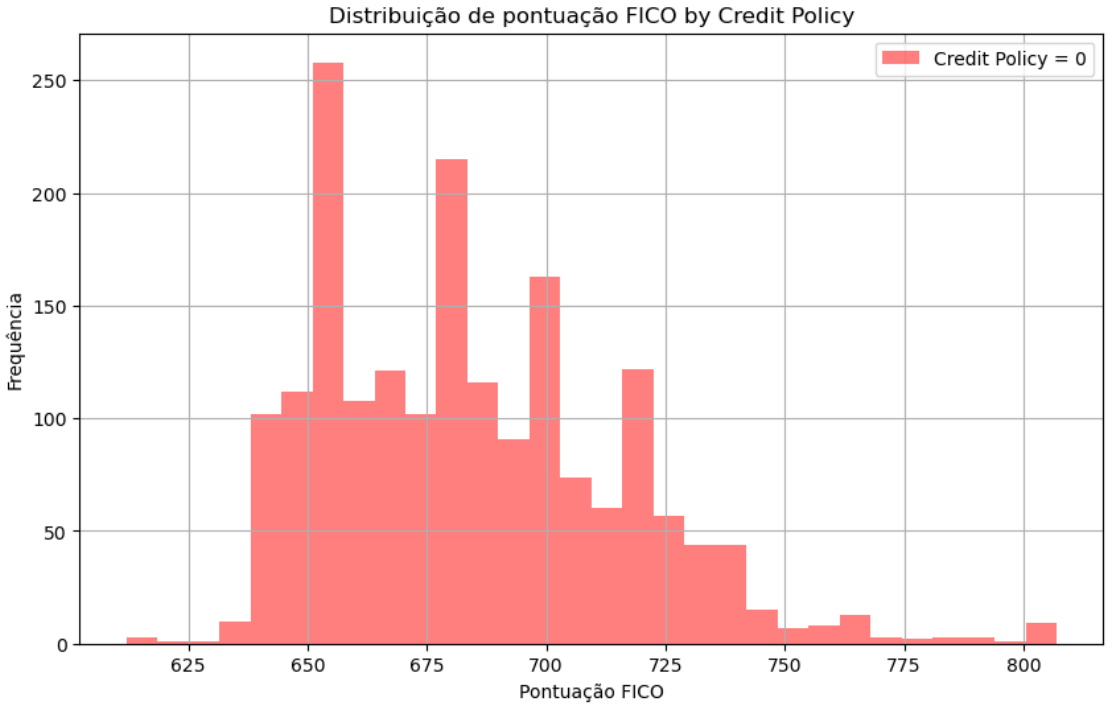
- Nota: Isto é bastante complicado, sinta-se à vontade para fazer referência às soluções. Você provavelmente precisará de uma linha de código para cada histograma, eu também recomendo usar o .hist() incorporado ao pandas. *

[7]:
```
plt.figure(figsize=(10, 6))
#plt.figure(figsize=(12, 8))

loans[loans['credit.policy'] == 1]['fico'].hist(alpha=0.5, color='blue',␣
 ↪bins=30, label='Credit Policy = 1')
#loans[loans['credit.policy'] == 0]['fico'].hist(alpha=0.5, color='red',␣
 ↪bins=30, label='Credit Policy = 0')

#plt.figure(figsize=(10, 6))
plt.legend()
plt.xlabel('Pontuação FICO')
plt.ylabel('Frequência')
plt.title('Distribuição de pontuação FICO by Credit Policy')
plt.show()
```



[8]:
```
plt.figure(figsize=(10, 6))
#plt.figure(figsize=(12, 8))
```

```
#loans[loans['credit.policy'] == 1]['fico'].hist(alpha=0.5, color='blue',␣
 ↪bins=30, label='Credit Policy = 1')
loans[loans['credit.policy'] == 0]['fico'].hist(alpha=0.5, color='red',␣
 ↪bins=30, label='Credit Policy = 0')

#plt.figure(figsize=(10, 6))
plt.legend()
plt.xlabel('Pontuação FICO')
plt.ylabel('Frequência')
plt.title('Distribuição de pontuação FICO by Credit Policy')
plt.show()
```



É possível ver que aqueles que atendem os critérios de empréstimos acabando tendo uma pontuação maior, provavelmente devido a melhores hábitos financeiros. Isso pode ser visto no código abaixo, onde quem atende os critérios tem uma pontuação média de 33 pontos a mais.

[9]:
```
# Média para 'Credit Policy = 1'
media_credit_policy_1 = loans[loans['credit.policy'] == 1]['fico'].mean()

# Média para 'Credit Policy = 0'
media_credit_policy_0 = loans[loans['credit.policy'] == 0]['fico'].mean()


Porcentagem = ((media_credit_policy_1/media_credit_policy_0)*100)-100
```

```
# Exibir as médias
print(f'Média FICO para Credit Policy = 1: {media_credit_policy_1}')
print(f'Média FICO para Credit Policy = 0: {media_credit_policy_0}')
print('A diferença entre eles é de: ', media_credit_policy_1 -␣
 ↪media_credit_policy_0)
print('A media de Credit Policy = 1 é ', Porcentagem, '% maior que a media de␣
 ↪Credit Policy = 0')
```

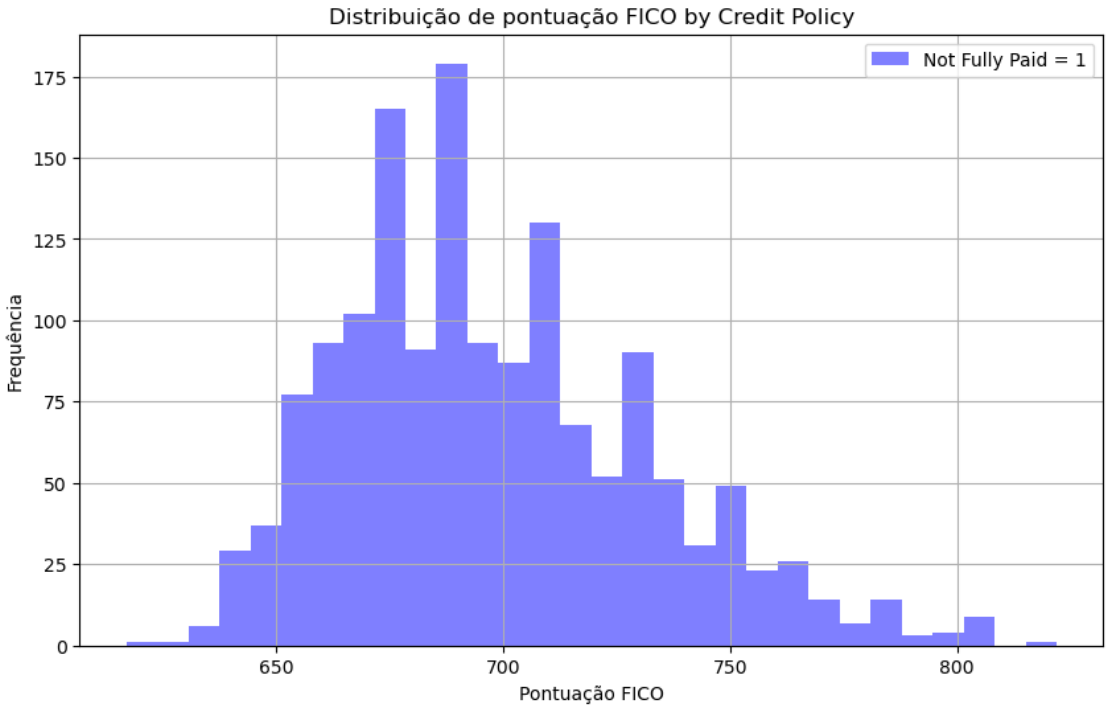
```
Média FICO para Credit Policy = 1: 717.3560311284047
Média FICO para Credit Policy = 0: 683.978051391863
A diferença entre eles é de:  33.37797973654165
A media de Credit Policy = 1 é  4.879978190618701 % maior que a media de Credit
Policy = 0
```

[10]:
```
# Total de registros
total_registros = len(loans)

# Contagem e porcentagem para 'Credit Policy = 1'
contagem_policy_1 = len(loans[loans['credit.policy'] == 1])
porcentagem_policy_1 = (contagem_policy_1 / total_registros) * 100

# Contagem e porcentagem para 'Credit Policy = 0'
contagem_policy_0 = len(loans[loans['credit.policy'] == 0])
porcentagem_policy_0 = (contagem_policy_0 / total_registros) * 100

# Exibir as porcentagens
print(f'Porcentagem de clientes com Credit Policy = 1: {porcentagem_policy_1:.
 ↪2f}%')
print(f'Porcentagem de clientes com Credit Policy = 0: {porcentagem_policy_0:.
 ↪2f}%')
```

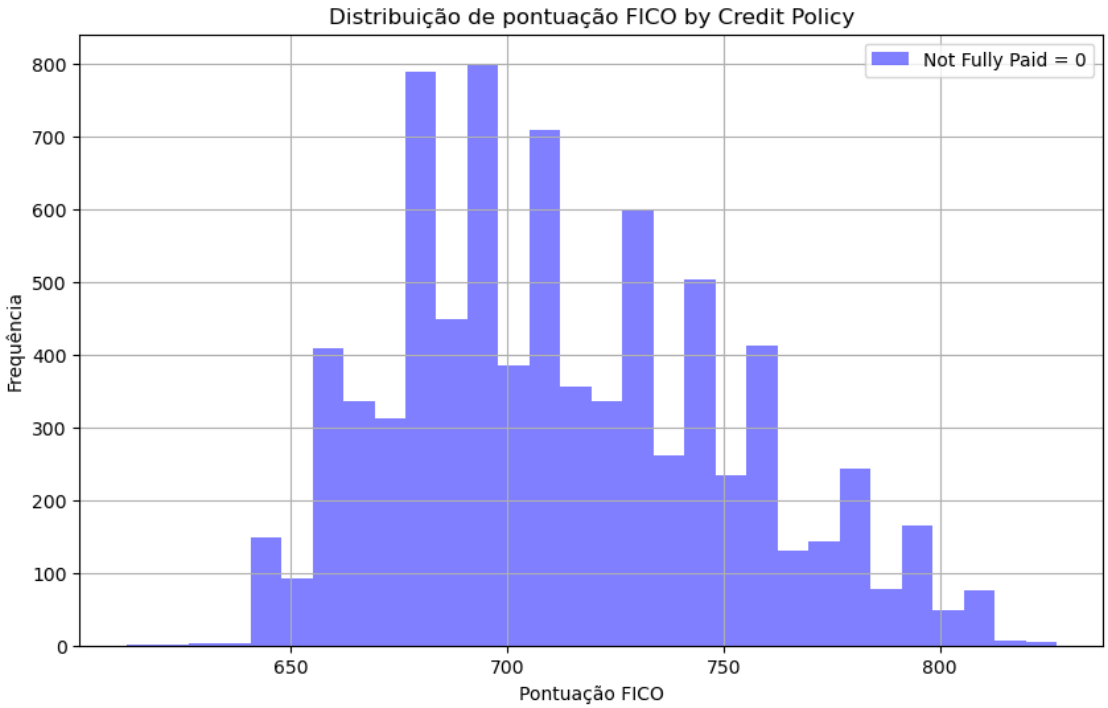
```
Porcentagem de clientes com Credit Policy = 1: 80.50%
Porcentagem de clientes com Credit Policy = 0: 19.50%
```

** Crie uma figura semelhante, mas dessa vez use a coluna not.fully.paid. **

[12]:
```
plt.figure(figsize=(10, 6))

loans[loans['not.fully.paid'] == 1]['fico'].hist(alpha=0.5, color='blue',␣
 ↪bins=30, label='Not Fully Paid = 1')

#plt.figure(figsize=(10, 6))
plt.legend()
plt.xlabel('Pontuação FICO')
plt.ylabel('Frequência')
plt.title('Distribuição de pontuação FICO by Credit Policy')
plt.show()
```

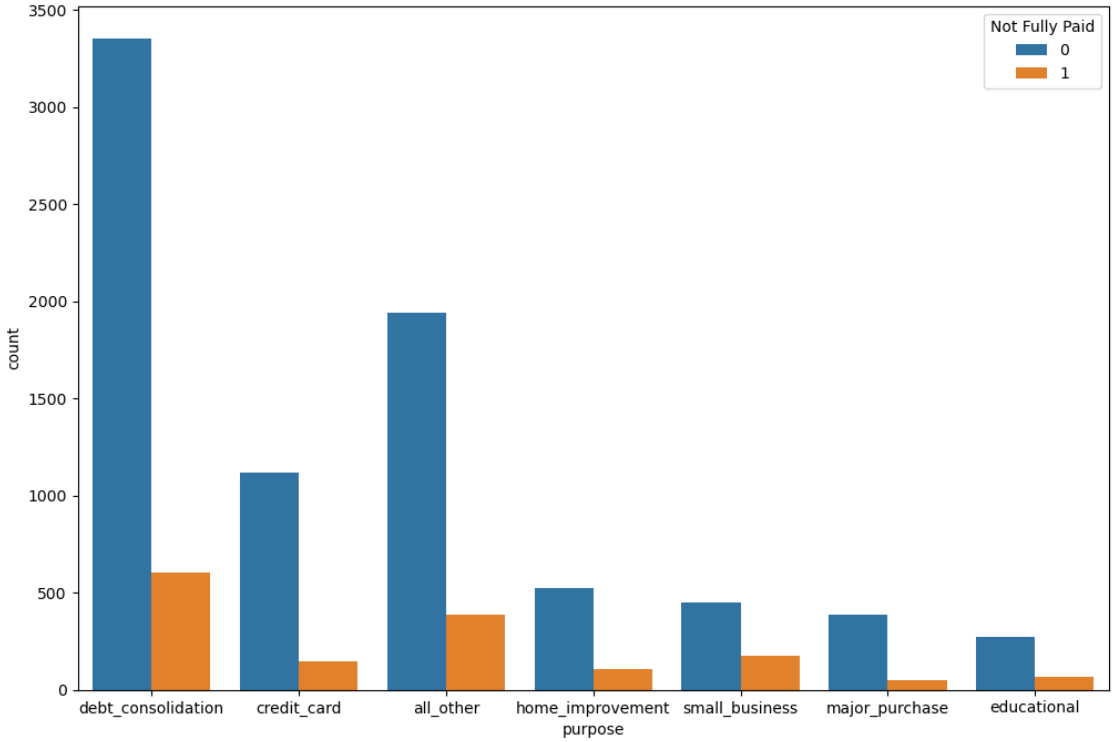


[13]:
```
plt.figure(figsize=(10, 6))

loans[loans['not.fully.paid'] == 0]['fico'].hist(alpha=0.5, color='blue',␣
 ↪bins=30, label='Not Fully Paid = 0')

#plt.figure(figsize=(10, 6))
plt.legend()
plt.xlabel('Pontuação FICO')
plt.ylabel('Frequência')
plt.title('Distribuição de pontuação FICO by Credit Policy')
plt.show()
```



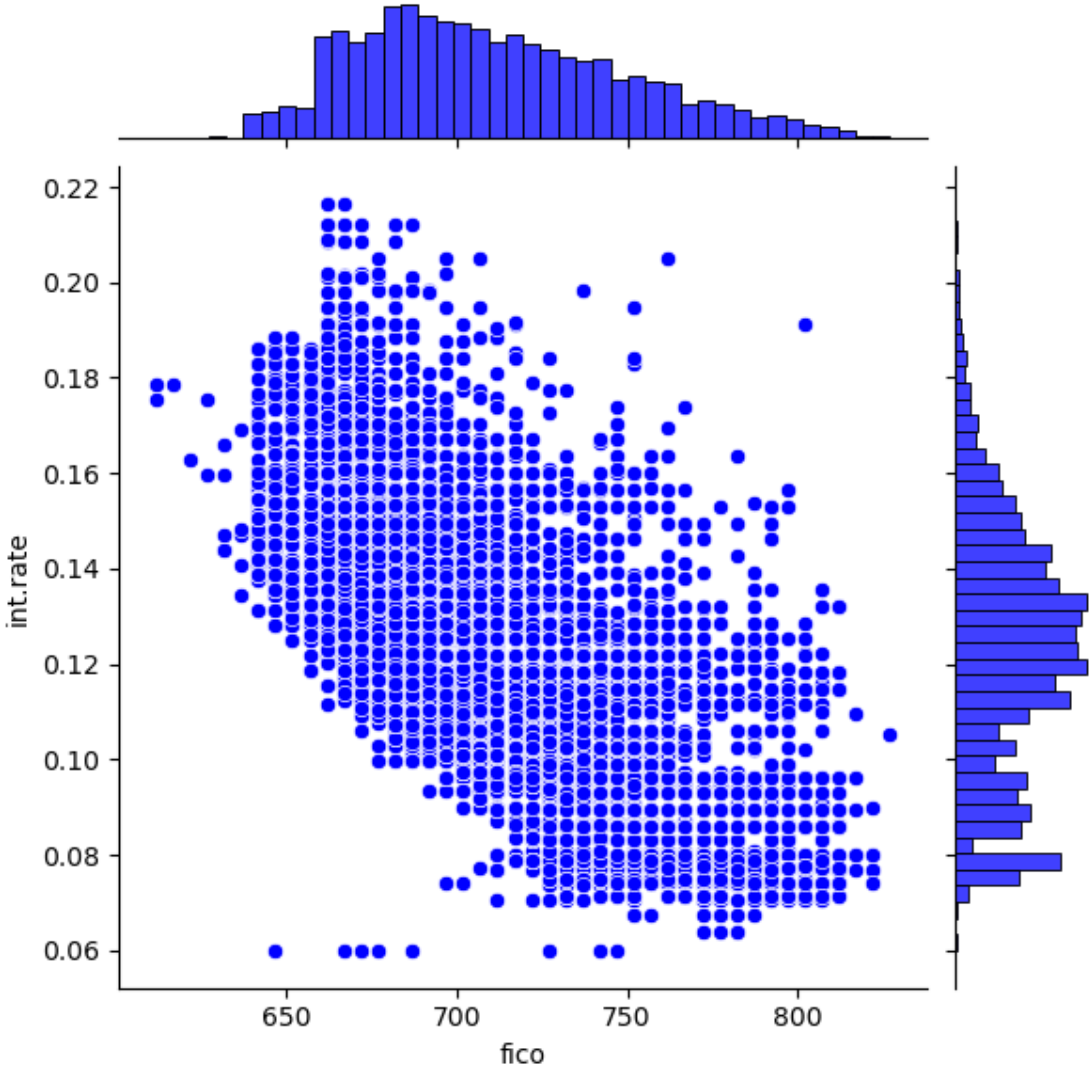
** Crie um countplot usando seaborn mostrando a contagens de empréstimos por finalidade, com a matiz de cor definido por not.fully.paid. **

[16]:
```
plt.figure(figsize=(12, 8))
countplot = sns.countplot(x='purpose', data=loans, hue='not.fully.paid')
plt.legend(title='Not Fully Paid')
plt.show()
```



** Veja a tendência entre o índice FICO e a taxa de juros. Recrie o seguinte jointplot. **

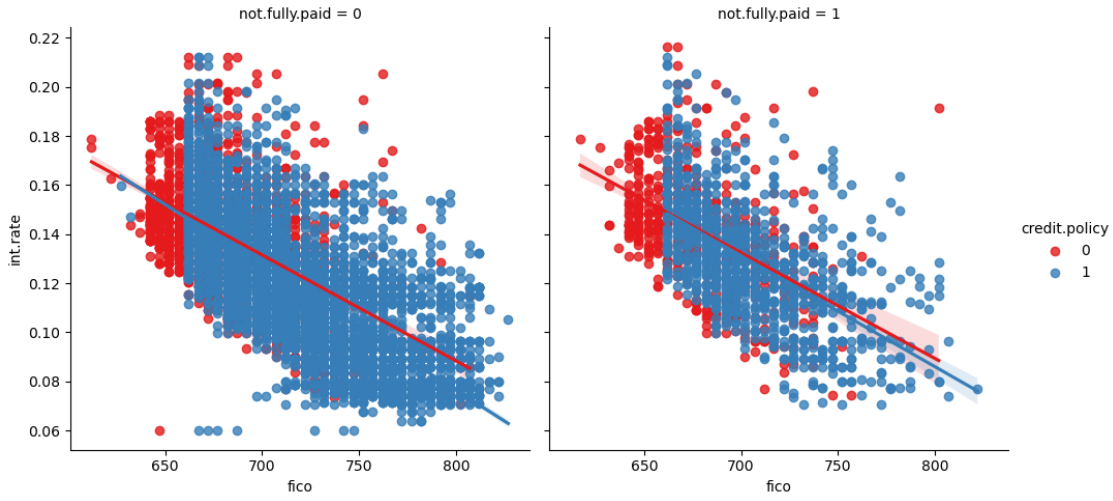
[18]:
```
jointplot = sns.jointplot(x='fico', y='int.rate', data=loans, color='blue')
```



As pessoas que tem um score menor acabam que, na média, pagam uma taxa de juros maior, refletindo uma prevenção contra a inadimplência.

** Crie os seguintes lmplots para ver se a tendência diferiu entre not.fully.paid e credit.policy. Verifique a documentação para lmplot() se você não consegue descobrir como separá-lo em colunas. **

[19]:
```
lmplots = sns.lmplot(x='fico', y='int.rate', data=loans, hue='credit.policy',␣
↪col='not.fully.paid', palette='Set1')
```



Mais uma vez provando que quanto maior for o score, menor a taxa de juros a se pagar. Mas mesmo aqueles que pagaram, os que não atendem a política de crédito tem uma taxa de juros mais alta do que aqueles que atendem, refletindo uma possível inadimplencia. Entre aqueles que não pagaram, a diferença entre a taxa é menos pronunciável, mas ainda existe.

4 Configurando os dados

Vamos nos preparar para configurar nossos dados para o nosso modelo de classificação de florestas aleatórias!

** Verifique loans.info() novamente. **

[20]:
```
loans.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
```

4.1 Recursos categóricos

Observe a coluna ** purpose ** como categórica.

Isso significa que precisamos transformá-los usando variáveis dummys para que Sklearn possa compreendê-las. Vamos fazer isso em um passo limpo usando pd.get_dummies.

Vamos mostrar uma maneira de lidar com essas colunas que podem ser expandidas para múltiplos parâmetros categóricos, se necessário.

** Crie uma lista de 1 elemento contendo a string 'purpose'. Chame esta lista de cat_feats. **

[21]:
```
cat_feats = ['purpose']
```

** Agora use "pd.get_dummies(loans, columns = cat_feats, drop_first = True)" para criar um DataFrame maior fixo que tenha novas colunas de recursos com variáveis dummy. Chame este dataframe de final_data. **

[22]:
```
final_data = pd.get_dummies(loans, columns=cat_feats, drop_first=True)
```

4.2 Divisão Treino-Teste de dados

Agora é hora de dividir nossos dados em um conjunto de treinamento e um conjunto de testes!

** Use sklearn para dividir seus dados em um conjunto de treinamento e um conjunto de testes como fizemos no passado. **

[29]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
```

[30]:
```
x = final_data.drop('not.fully.paid', axis=1)
y = final_data['not.fully.paid']
```

4.3 Training a Decision Tree Model

Vamos começar treinando uma única árvore de decisão primeiro!

** Import DecisionTreeClassifier **

[31]:
```
from sklearn.tree import DecisionTreeClassifier
```

** Crie uma instância de DecisionTreeClassifier() chamada dtree e fite-a com os dados de treinamento. **

[32]:
```
DecisionTreeClassifier = DecisionTreeClassifier()
```

[33]:
```
DecisionTreeClassifier.fit(x, y)
```

[33]:
```
DecisionTreeClassifier()
```

4.4 Previsões e avaliação da árvore de decisão

** Faça previsões do conjunto de teste e crie um relatório de classificação e uma matriz de confusão. **

[34]:
```
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=42)
```

[35]:
```
DecisionTreeClassifier.fit(X_train, y_train)
predictions = DecisionTreeClassifier.predict(X_test)
```

[36]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      2408
           1       0.20      0.22      0.21       466

    accuracy                           0.73      2874
   macro avg       0.52      0.52      0.52      2874
weighted avg       0.74      0.73      0.74      2874
```

[37]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[1998  410]
 [ 364  102]]
```

4.5 Treinando o modelo de florestas aleatórias

Agora é hora de treinar nosso modelo!

** Crie uma instância da classe RandomForestClassifier e ajuste-a aos nossos dados de treinamento da etapa anterior. **

[42]:
```
from sklearn.ensemble import RandomForestClassifier
RandomForestClassifier = RandomForestClassifier()
```

[43]:
```
RandomForestClassifier.fit(X_train, y_train)
```

[43]:
```
RandomForestClassifier()
```

[27]:
```

```

[27]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

4.6 Previsões e Avaliação

Vamos prever os valores do y_test e avaliar o nosso modelo.

** Preveja a classe de not.fully.paid para os dados X_test. **

[44]:
```
rf_predictions = RandomForestClassifier.predict(X_test)
```

** Agora crie um relatório de classificação dos resultados. Você recebe algo estranho ou algum tipo de aviso? **

[48]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

[46]:
```
print(classification_report(y_test, rf_predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91      2408
           1       0.30      0.02      0.04       466

    accuracy                           0.83      2874
   macro avg       0.57      0.51      0.47      2874
weighted avg       0.75      0.83      0.77      2874
```

** Mostre a Matriz de Confusão para as previsões. **

[49]:
```
print(confusion_matrix(y_test, rf_predictions))
```

```
[[2387   21]
 [ 457    9]]
```

** O que performou melhor: a floresta aleatória ou a árvore de decisão? **

Comparando ambos fica claro que a floresta aleatória teve uma performance superior, tendo uma acurácia melhor e médias superiores.

[50]:
```
# Relatório de classificação para a árvore de decisão
print("Relatório de classificação para a Árvore de Decisão:")
print(classification_report(y_test, predictions))

# Relatório de classificação para a floresta aleatória
print("Relatório de classificação para a Floresta Aleatória:")
print(classification_report(y_test, rf_predictions))
```

```
Relatório de classificação para a Árvore de Decisão:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      2408
           1       0.20      0.22      0.21       466

    accuracy                           0.73      2874
   macro avg       0.52      0.52      0.52      2874
weighted avg       0.74      0.73      0.74      2874

Relatório de classificação para a Floresta Aleatória:
              precision    recall  f1-score   support

           0       0.84      0.99      0.91      2408
           1       0.30      0.02      0.04       466

    accuracy                           0.83      2874
   macro avg       0.57      0.51      0.47      2874
weighted avg       0.75      0.83      0.77      2874
```

[51]:
```
print('Matriz de confusão para a Árvore de Decisão:')
print(confusion_matrix(y_test, predictions))

print('Matriz de confusão para a Floresta Aleatória:')
print(confusion_matrix(y_test, rf_predictions))
```

```
Matriz de confusão para a Árvore de Decisão:
[[1998  410]
 [ 364  102]]
Matriz de confusão para a Floresta Aleatória:
[[2387   21]
 [ 457    9]]
```