# Hitelképsesség megállapítása

Adatset szerkezete:

|  |  |
| --- | --- |
| **Variable** | Leírás, megjegyzés |
| **Loan\_ID** | Egyedi hitel azonosító |
| **Gender** | Nem (Male/Female) |
| **Married** | Az igénylő házas-e? (Y/N) |
| **Dependents** | Eltartottak száma |
| **Education** | Igénylő iskolázottsága, végzettsége (Graduate/ Under Graduate) |
| **Self\_Employed** | Önfoglalkoztatott-e? (Y/N) |
| **ApplicantIncome** | Igénylő havi bevétele |
| **CoapplicantIncome** | Igénylőtárs bevétele |
| **LoanAmount** | Hitelösszeg (ezer $) |
| **Loan\_Amount\_Term** | A hitel futamideje hónapokban |
| **Credit\_History** | Hiteltörténet (1 ha van, 0, ha nincs) |
| **Property\_Area** | Ingatlan lakókörnyezete: Urban/ Semi Urban/ Rural |
| **Loan\_Status** | Hitelkérelmet elfogadták-e? (0/1) |

Szükséges könyvtárak importálása és a dataset betöltése:

import pandas as pd

import numpy as np

import matplotlib as plt

df = pd.read\_csv("loan.csv")

Adatok gyors áttekintése (első 10 sor):

df.head(10)

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | 360.0 | 1.0 | Urban | Y |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 | Rural | N |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | 360.0 | 1.0 | Urban | Y |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 | Urban | Y |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | 360.0 | 1.0 | Urban | Y |
| **5** | LP001011 | Male | Yes | 2 | Graduate | Yes | 5417 | 4196.0 | 267.0 | 360.0 | 1.0 | Urban | Y |
| **6** | LP001013 | Male | Yes | 0 | Not Graduate | No | 2333 | 1516.0 | 95.0 | 360.0 | 1.0 | Urban | Y |
| **7** | LP001014 | Male | Yes | 3+ | Graduate | No | 3036 | 2504.0 | 158.0 | 360.0 | 0.0 | Semiurban | N |
| **8** | LP001018 | Male | Yes | 2 | Graduate | No | 4006 | 1526.0 | 168.0 | 360.0 | 1.0 | Urban | Y |
| **9** | LP001020 | Male | Yes | 1 | Graduate | No | 12841 | 10968.0 | 349.0 | 360.0 | 1.0 | Semiurban | N |

A számadatokat tartalmazó adatokról némi statisztikai információ:

df.describe()

|  | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** |
| --- | --- | --- | --- | --- | --- |
| **count** | 614.000000 | 614.000000 | 592.000000 | 600.00000 | 564.000000 |
| **mean** | 5403.459283 | 1621.245798 | 146.412162 | 342.00000 | 0.842199 |
| **std** | 6109.041673 | 2926.248369 | 85.587325 | 65.12041 | 0.364878 |
| **min** | 150.000000 | 0.000000 | 9.000000 | 12.00000 | 0.000000 |
| **25%** | 2877.500000 | 0.000000 | 100.000000 | 360.00000 | 1.000000 |
| **50%** | 3812.500000 | 1188.500000 | 128.000000 | 360.00000 | 1.000000 |
| **75%** | 5795.000000 | 2297.250000 | 168.000000 | 360.00000 | 1.000000 |
| **max** | 81000.000000 | 41667.000000 | 700.000000 | 480.00000 | 1.000000 |

Amit rögtön észre vehetünk:

1. LoanAmount (614 – 592) 22 hiányzó érték.
2. Loan\_Amount\_Term (614 – 600) 14 hiányzó érték.
3. Credit\_History (614 – 564) 50 hiányzó érték.
4. A hitelfelvevők 84%-nak van hiteltörténete (a credit history átlaga 0,84).
5. Úgy tűnik, hogy a hitel pályázó jövedelmének (ApplicantIncome) eloszlása megfelel a várakozásoknak. Hasonlóan ugyanez igaz a CoapplicantIncome-mezőre.

A nem numerikus értékek (például Property\_Area stb.) frekvencia eloszlását is érdemes áttekinteni annak megértéséhez, hogy értelme van-e vagy sem. Számoljuk meg, hogy az ingatlan lakókörnyezet értékeknek milyen előfordulásai vannak:

df['Property\_Area'].value\_counts()

Semiurban 233

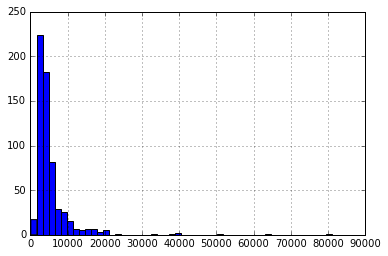
Urban 202

Rural 179

Name: Property\_Area, dtype: int64

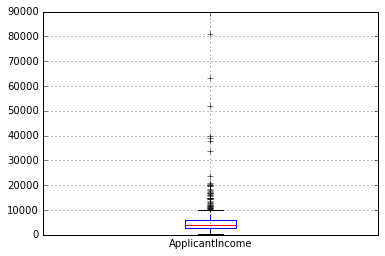
**Eloszláselemzés**

df['ApplicantIncome'].hist(bins=50)



Megfigyelhetjük, hogy van néhány extrém érték. Ezért is kellett 50 kategória. Ahhoz, hogy megértsük az eloszlást, dolgozzunk tovább ún blox plot diagrammal:

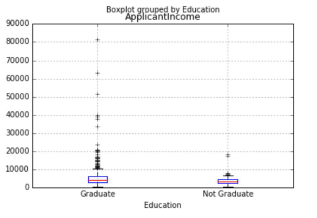
df.boxplot(column='ApplicantIncome')



Ez megerősítheti azt a feltevésünket, hogy több extrém érték (outlier) jövedelmi érték is lehet. Mindez a társadalom jövedelmi egyenlőtlenségeinek tulajdonítható be.

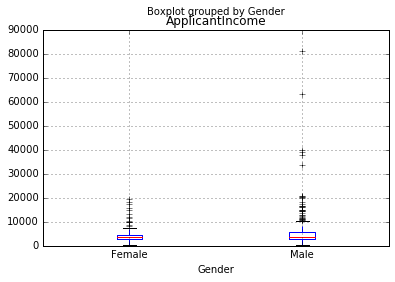
Ahhoz, hogy a különböző paraméterek hatásait is tudjuk vizsgálni bontsuk szét az adatsort nem, iskolai végzettség szerint:

df.boxplot(column='ApplicantIncome', by = 'Education')



illetve:

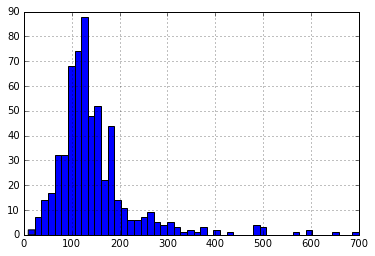
df.boxplot(column='ApplicantIncome', by = 'Gender')



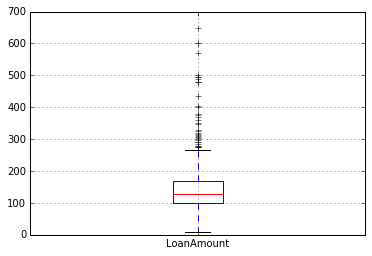
Látható, hogy a diplomások többet keresnek, ugyanakkor van közöttük néhány nagyon kiugró érték (outlier).

Nézzük meg a hitelösszeg eloszlását is (hisztogrammal és boxplottal):

df['LoanAmount'].hist(bins=50)

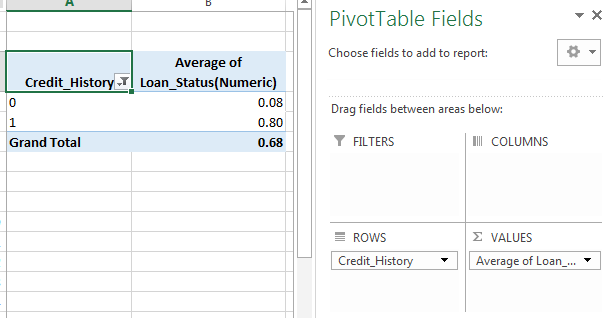


df.boxplot(column='LoanAmount')



Látható, hogy itt is van néhány kiugró érték. Haladjunk picit tovább, nézzük meg, hogy milyen hatása van a hiteltörténetnek. Ezt Excel pivot táblával is meg lehetne vizsgálni?

Mivel a Hitel státusz értéke 0 vagy 1 lehet, ezért ennek az átlaga mutathatja hitelkérelem esélyét.

Kapcsolódó Excel Pivot tábla:  


Pandas Pivot tábla referencia: <http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.pivot_table.html#pandas.DataFrame.pivot_table>

temp1 = df['Credit\_History'].value\_counts(ascending=True)

temp2 = df.pivot\_table(values='Loan\_Status',index=['Credit\_History'],aggfunc=lambda x: x.map({'Y':1,'N':0}).mean())

print ('A hiteltörténet gyakorisági eloszlása:')

print (temp1)

print ('\nAz egyes osztályok milyen valószínűséggel kapnak hitelt:')

print (temp2)

A hiteltörténet gyakorisági eloszlása:

0.0 89

1.0 475

Name: Credit\_History, dtype: int64

Az egyes osztályok milyen valószínűséggel kapnak hitelt:

Loan\_Status

Credit\_History

0.0 0.078652

1.0 0.795789

Rajzoljunk egy Excelhez hasonló Pivot táblát:

fig = plt.figure(figsize=(8,4))

ax1 = fig.add\_subplot(121)

ax1.set\_xlabel('Hiteltörténet')

ax1.set\_ylabel('Igénylök száma')

ax1.set\_title("Igénylök Hiteltörténet szerint")

temp1.plot(kind='bar')

ax2 = fig.add\_subplot(122)

temp2.plot(kind = 'bar')

ax2.set\_xlabel('Hiteltörténet')

ax2.set\_ylabel('Hitel valószínüsége')

ax2.set\_title("Hitel valószínüsége hiteltörténet szerint")

A képen képernyőkép látható

A leírás teljesen megbízhatóA képen képernyőkép látható

A leírás teljesen megbízható

**Kitekintés:**

Matplotlib Pyplot tutorial: <https://matplotlib.org/users/pyplot_tutorial.html>

Malplotlib grafikonok listája: <https://matplotlib.org/gallery.html>

Két nagyon hasznos portál az egyes diagrammtípusok megértéséhez:

<https://datavizcatalogue.com/>

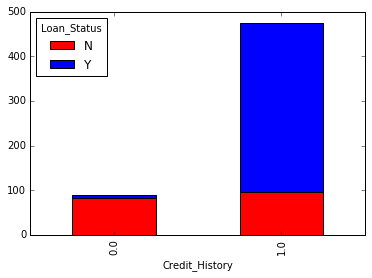
<http://datavizproject.com/>

Látható, hogy a hiteltörténettel rendelkezők közel 8-szor nagyobb eséllyel kapnak újra hitelt Hasonlóan vizsgálhatunk más paramétereket is: családi állapot, lakókörnyezet, stb.

Halmozott oszlopdiagrammal akár össze is vonhatjuk a kettőt:

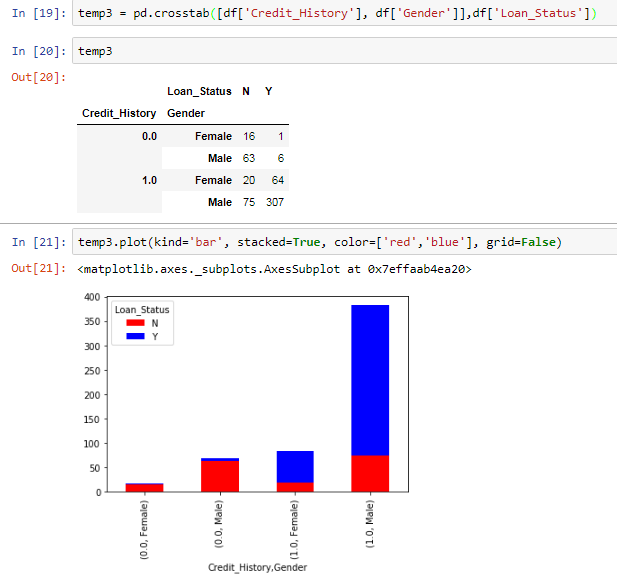
temp3 = pd.crosstab(df['Credit\_History'], df['Loan\_Status'])

temp3.plot(kind='bar', stacked=**True**, color=['red','blue'], grid=**False**)



Az elemzésbe a Pivot táblához hasonlóan belevehetjük a nem mezőt (Gender) is (a crosstab <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.crosstab.html> – kereszttábla segítségével – ez utóbbi Accessből lehet még ismerős):

temp3 = pd.crosstab([df['Credit\_History'], df['Gender']],df['Loan\_Status'])



Az ábrázolásnál álljunk meg egy kicsit és nézzük meg, hogy miért az oszlopdiagram volt a jó választás most, nézzük meg a plot() függvényt (<https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html#pandas.DataFrame.plot>):

Próbáljuk ki:

temp3.plot()

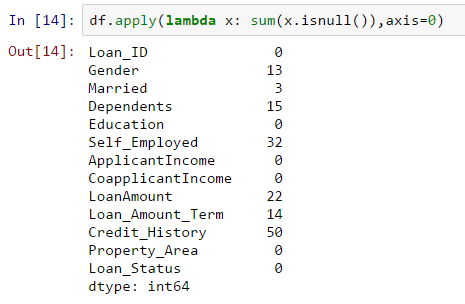
temp3.plot(kind= 'barh') #hist, box, area,... Melyik mit mond, mit jelent?

**Hiányzó értékek elemzése**

Egy nagyon jó cikk: <https://www.analyticsvidhya.com/blog/2014/09/data-munging-python-using-pandas-baby-steps-python/>

Az alábbi parancsban az isnull() 1-el tér vissza az egyes oszlopokban, ha az oszlop értéke null

df.apply(**lambda** x: sum(x.isnull()),axis=0)



Bár a hiányzó értékek nem nagyon magas számban vannak, de számos változónak esetén előfordulnak, és mindegyiküket meg kell becsülni és hozzá kell adni az adatokhoz. Részletesebb leírás itt található az alkalmazható technikákról: <https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/>

Nem mindig a NaN jelent hiányzó értéket, lásd a Loan\_Amount\_Term=0 esetet!

Számos módon lehet kitölteni a hitel összegének hiányzó értékeit - a legegyszerűbb lenne a helyettesítés átlaggal (de ezt most ne csináljuk meg): df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)

A legjobb módszer a supervised learning modell felépítése lenne a hitel összegének más változók alapján történő előrejelzésére (erről később).

Nézzünk meg egy egyszerűbb, de nem túl egyszerű lehetőséget:

Azt feltételezhetjük, hogy a végzettségnek és annak, hogy valaki vállalkozó-e, nagy hatása van a hitel összegére (ezt egy diagrammal érdemes is leellenőrizni).

Mint korábban láttuk, az Self\_Employednek van néhány hiányzó értéke. Nézzük meg a frekvencia táblázatot:

df['Self\_Employed'].value\_counts()

No 500

Yes 82

Name: Self\_Employed, dtype: int64

Mivel az értékek ~ 86% -a "Nem", a hiányzó értékeket "No" -ként lehet bevinni, nagy valószínűséggel. Ezt a következő kóddal lehet elvégezni:

df['Self\_Employed'].fillna('No',inplace=True)

Nézzük meg az iskolázottság és a vállalkozó/beosztottság közötti összefüggéseket, a egy pivot tábla segítségével nézzük meg a hitelösszegek mediánját az egyes csoportokban (miért? <https://kiszamolo.hu/a-median-jelentese/>).

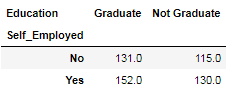
table = df.pivot\_table(values='LoanAmount',

index='Self\_Employed',

columns='Education',

aggfunc=np.median)

table



Melyik sorokban szerepel üres / hiányzó érték?

idx = df.loc[df['LoanAmount'].isnull(), ['Self\_Employed','Education', 'LoanAmount']].index

idx

Majd az előző medián kalkuláció alapján töltsük ki a hiányzó értékeket:

df = df.set\_index(['Education','Self\_Employed'])

df['LoanAmount'].fillna(table.unstack(), inplace=True)

df = df.reset\_index()

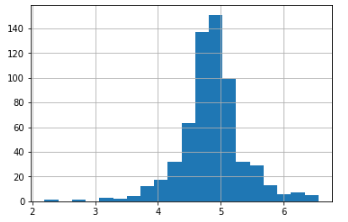
Ellenőrizzük is le, hogy melyik, korábban üres mezőket cseréltük le:

print (df.loc[df.index.isin(idx), ['Self\_Employed','Education', 'LoanAmount']])

Gyakori probléma, hogy a szélsőségesen kiugró értékek torzítják az elemzéseinket. Ezeket mindenképpen meg kell vizsgálnunk, számos esetben tekinthetjük ezeket hibás értékeknek, ún. outliereknek, s eldobhatjuk őket. Elsőre úgy tűnik, hogy a hitelösszegben is vannak ilyenek, de miért ne lehetne, hogy valaki, a többiekhez képest kiugrónak látszó összegű hitelt igényeljen? Tompítsuk tehát azok hatását. Inkább, minthogy kitörölnénk:

df['LoanAmount\_log'] = np.log(df['LoanAmount'])

df['LoanAmount\_log'].hist(bins=20)



Most az eloszlás sokkal közelebb áll a normálhoz, és a szélsőséges értékek hatása jelentősen lecsökkent.

Hasonlóan érdekes a háztartás bevételeit is vizsgálni, sőt a két mezőt összevonni:

df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']

df['TotalIncome\_log'] = np.log(df['TotalIncome'])

df['LoanAmount\_log'].hist(bins=20)

Hasonlóan kell elvégezni a többi hiányzó mező kitöltését is (Gender, Married, Dependents, Loan\_Amount\_Term, Credit\_History).

Érdemes azon is elgondolkodni, hogy mi jöhet még ki az adatokból? Például ha készítünk egy oszlopot, hogy az igényelt hitel és a teljes bevétel arányosításával (LoanAmount/TotalIncome), akkor kapunk egy kis, egyszerű elképzelést arról, hogy a hiteligénylő vissza tudja-e fizetni a hitelét.

# Prediktív modell készítése

Erre a célra az egyik leggyakrabban használt Python library a Skicit-Learn (sklearn).

További infók:

<http://scikit-learn.org/>

<http://scikit-learn.org/stable/tutorial/basic/tutorial.html>

<https://www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machine-learning-tool/>

Mivel a sklearn-ben minden bemenetnek numerikusnak kell lennie, az összes kategorikus (object típusú) változót számszerűsíteni kell a kategóriák kódolásával. Ezt a következő kóddal lehet elvégezni:

from sklearn.preprocessing import LabelEncoder

var\_mod = ['Married','Dependents','Education','Self\_Employed','Property\_Area','Loan\_ID']

le = LabelEncoder()

for i in var\_mod:

df[i] = le.fit\_transform(df[i].astype('str'))

df.dtypes

Megjegyés: vagy string, vagy int konverzióval (df[i] = le.fit\_transform(df[i]))

Érdemes lehet előtte megnézni az adattípusokat: df.dtypes

Importáljuk a szükséges modulokat. A következőkben meghatároznunk egy generikus osztályozási (Generic Classification) függvényt, amely bemeneti modellként veszi fel a modellt, és meghatározza a pontosságot (Accuracy) és a keresztellenőrzés (Cross-Validation) eredményeit.

További információk az egyes machine learning algoritmusokról ebben a nagyon jó bevezető cikkben találhatóak: <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>

#A scikit learn modulból a modellek importálása:

from sklearn.linear\_model import LogisticRegression

from sklearn.cross\_validation import KFold #K-fold cross validation

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

from sklearn import metrics

# Általános funkció a klasszifikációs modell elkészítéséhez és a teljesítmény eléréséhez:

def classification\_model(model, data, predictors, outcome):

#Modell illesztése:

model.fit(data[predictors],data[outcome])

#Előrejelzése készítése (prediction) a tréning adatszeten:

predictions = model.predict(data[predictors])

#Pontosság (Accuracy) kiírása

accuracy = metrics.accuracy\_score(predictions,data[outcome])

print("Accuracy :{0:.3%}".format(accuracy))

#K-fold cross-validation végrehajtása 5 hajtással

kf = KFold(data.shape[0], n\_folds=5)

error = []

for train, test in kf:

# A tanuló adatok szűrése

train\_predictors = (data[predictors].iloc[train,:])

# Az a cél, amelyet az algoritmus tanítására használunk.

train\_target = data[outcome].iloc[train]

# Az algoritmus tanítása az előrejelzőkkel és célokkal.

   model.fit(train\_predictors, train\_target)

#Hibák rögzítése a cross-validation futásakor

   error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

print("Cross-Validation Score :{0:.3%}". format(np.mean(error)))

#Illeszzük újra úgy a modellt:

model.fit(data[predictors],data[outcome])

## Lineáris Regresszió

Bevezetés a Lineráis Regresszióról: <https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/>

Könnyen be tudunk vezetni egy kis intuitív hipotézist. A hitelszerzés esélye magasabb lesz az alábbi esetekben:

* A hiteligénylőknek volt már credit history-juk
* A kérelmezők és hozzátartozójuk magasabb bevétellel rendelkezik
* Magasabb iskolai végzettségük van
* Nagyobb növekedési perspektívával rendelkező városi övezetben élnek.

Készítsük el az első modellünket a "Credit\_History" segítségével.

outcome\_var = 'Loan\_Status'

model = LogisticRegression()

predictor\_var = ['Credit\_History']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 80.945%

Cross-Validation Score : 80.946%

Egyéb változók kombinálásával:

predictor\_var = ['Credit\_History','Education','Married','Self\_Employed','Property\_Area']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 80.945%

Cross-Validation Score : 80.946%

Várakozásaink szerint a pontosság növelhető a változók hozzáadásával. De ez csak látszólag segít, nem lehet mindent hozzáadni, vannak fontosabb és kevésbé fontos (sőt) változók. Nézzünk egy jobb modellezési technikát, a döntési fákat (Decision Tree). Egy másik lehetőség lenne a Feature Engineering.

## Decision Tree

Egy másik predikciós modell, a lineáris regresszióhoz képest nagyobb pontosságot érhetünk el vele. További információk például itt: <https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/>

model = DecisionTreeClassifier()

predictor\_var = ['Credit\_History','Gender','Married','Education']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 81.930%

Cross-Validation Score : 76.656%

Itt a kategorikus változókon alapuló modellnek nincs hatása, mivel a Hiteltörténet a domináns. Próbáljunk néhány numerikus változót:

predictor\_var = ['Credit\_History','Loan\_Amount\_Term','LoanAmount\_log']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 92.345%

Cross-Validation Score : 71.009%

Megfigyelhetjük, hogy bár a pontosság növelte a változók hozzáadását, a kereszt-érvényesítési hiba lelassult. Ez a modell túlzottan illeszkedő modelljének eredménye. Próbáljunk ki egy még kifinomultabb algoritmust, és nézzük meg, segít-e:

## Random Forest

Egy érdekes bevezető: <https://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/>

A Random Forest előnye, hogy minden jellemzővel együtt tudunk működni, és visszaad egy ún. jellemző fontossági mátrixot, amelyet a jellemzők kiválasztására használhatunk.

model = RandomForestClassifier(n\_estimators=100)

predictor\_var = ['Gender', 'Married', 'Dependents', 'Education',

'Self\_Employed', 'Loan\_Amount\_Term', 'Credit\_History', 'Property\_Area',

'LoanAmount\_log','TotalIncome\_log']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 100.000%

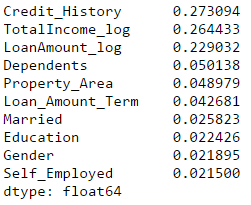
Cross-Validation Score : 78.179%

Látjuk, hogy a pontosság 100% a tréning szetben. Ez bizony egy ún. túlfutás és kétféleképpen oldható meg:

* A prediktorok számának csökkentésével
* A modell paramétereinek hangolásával

featimp = pd.Series(model.feature\_importances\_, index=predictor\_var).sort\_values(ascending=False)

print featimp



Használjuk a top 5 változót a modell létrehozásához. A Random Forest modellek paramétereit is egy kicsit megváltoztatjuk:

model = RandomForestClassifier(n\_estimators=25, min\_samples\_split=25, max\_depth=7, max\_features=1)

predictor\_var = ['TotalIncome\_log','LoanAmount\_log','Credit\_History','Dependents','Property\_Area']

classification\_model(model, df,predictor\_var,outcome\_var)

Accuracy : 82.899%

Cross-Validation Score : 81.461%

Vegyük észre, hogy bár a pontosság csökken, de a kereszt-érvényesítési pontszám javul, ami azt mutatja, hogy a modell jól általánosítható. Ne feledjük el, hogy a Random Forest modellek nem pontosan megismételhetők. Különböző futások kisebb eltéréseket eredményeznek a randomizálás miatt. De a kimenetnek a közel hasonlónak kell maradnia.

Amúgy a Lineáris Regresszióhoz képest a Cross-Validation csak picit változott.

Egy kifinomultabb modell használata nem garantálja a jobb eredményeket. Érdemes kerülni a komplex modellezési technikák használatát fekete dobozként anélkül, hogy megértenénk az alapfogalmakat.

Ami segíthet még, az az ún Feature Engineering:

<https://medium.com/mindorks/what-is-feature-engineering-for-machine-learning-d8ba3158d97a>

<https://www.datacamp.com/community/tutorials/feature-engineering-kaggle>