Overfitting and Cross validation

Part 2

- Introduction -

Combinations and Iterations

INTRODUCTION

- The number of different groups of *k* objects that can be selected from a set of *n* objects is equal to

$$ {}_nC_k = \frac{n!}{k!(n-k)!} = \binom{n}{k} $$

- It is the number of combinations of *n* objects taken *k* at a time. This number is referred to as *n* choose *k*
- For example

The combinations of {1,2,3,4} taken k=2 at a time are

{1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}

there are 6 = 4! / [(2!)(4-2) !] combinations

EXAMPLE 2

.

```
import numpy as np
import pandas as pd
```

```
# library for combinatorial numbers

from scipy.special import comb
```

```
# library for combinations

import itertools
```

```
# Examples
```

```
# 7 choose 3
comb(7,3)
```

```
35.0
```

EXAMPLE 2

.

```
list1 = ['a','b','c','d']
n = len(list1)
n
```

```
4
```

```
# Number of sets with two elements
comb(4,2)
```

```
6.0
```

```
# all possible sets with two elements
list(itertools.combinations(list1,2))
```

```
[('a', 'b'), ('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd'), ('c', 'd')]
```

EXAMPLE 2

.

```
list1 = ['a','b','c','d']
n = len(list1)
n
```

```
4
```

```
# Number of sets with 3 elements
comb(4,3)
```

```
4.0
```

```
# all possible sets with 3 elements
list(itertools.combinations(list1,3))
```

```
[('a', 'b', 'c'), ('a', 'b', 'd'), ('a', 'c', 'd'), ('b', 'c', 'd')] )]
```

EXAMPLE 2 – feature-cv7.ipynb

```
df = pd.read_csv('Credit.csv')
df[:5]
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331

```
df = df.loc[:,df.dtypes != object]
df[:5]
```

← select numerical columns only

	Income	Limit	Rating	Cards	Age	Education	Balance
0	14.891	3606	283	2	34	11	333
1	106.025	6645	483	3	82	15	903
2	104.593	7075	514	4	71	11	580
3	148.924	9504	681	3	36	11	964
4	55.882	4897	357	2	68	16	331

EXAMPLE 2

```
list(df.columns)
```

```
['Income', 'Limit', 'Rating', 'Cards', 'Age', 'Education', 'Balance']
```

```
n = len(df.columns)
n
```

```
7
```

```
# Number of sets with two column names
comb(7,2)
```

```
21.0
```

```
# all sets of two column names
list(itertools.combinations(df.columns,2))
```

```
[('Income', 'Limit'),
 ('Income', 'Rating'),
 ('Income', 'Cards'),
 ('Income', 'Age'),
 ('Income', 'Education'),
 ('Income', 'Balance'),
 ('Limit', 'Rating'),
 ('Limit', 'Cards'),
 ('Limit', 'Age'),
 ('Limit', 'Education'),
 ('Limit', 'Balance'),
 ('Rating', 'Cards'),
 ('Rating', 'Age'),
 ('Rating', 'Education'),
 ('Rating', 'Balance'),
 ('Cards', 'Age'),
 ('Cards', 'Education'),
 ('Cards', 'Balance'),
 ('Age', 'Education'),
 ('Age', 'Balance'),
 ('Education', 'Balance')]
```

EXAMPLE 2

.

```
# Number of sets with 6 column names
comb(7,6)
```

```
7.0
```

```
# all sets of 6 column names
list(itertools.combinations(df.columns,6))
```

```
[('Income', 'Limit', 'Rating', 'Cards', 'Age', 'Education'),
 ('Income', 'Limit', 'Rating', 'Cards', 'Age', 'Balance'),
 ('Income', 'Limit', 'Rating', 'Cards', 'Education', 'Balance'),
 ('Income', 'Limit', 'Rating', 'Age', 'Education', 'Balance'),
 ('Income', 'Limit', 'Cards', 'Age', 'Education', 'Balance'),
 ('Income', 'Rating', 'Cards', 'Age', 'Education', 'Balance'),
 ('Limit', 'Rating', 'Cards', 'Age', 'Education', 'Balance')]
```

Example 2
Best Model for Prediction

EXAMPLE 2

The *Credit.csv* file contains credit information from 400 customers

- Balance (average credit card debt) ← response
- Age
- Cards (number of credit cards)
- Education (years of education)
- Income (in thousands of dollars)
- Limit (credit limit)
- Rating (credit rating)
- Gender
- Student (student status)
- Married (marital status)
- Ethnicity (Caucasian, African American or Asian)

Find the best model to predict the customer Balance using adj-R^2 and AIC

BEST REGRESSION MODEL

```
import numpy as np
import pandas as pd
```

```
import itertools
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
from sklearn.model_selection import train_test_split

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

BEST REGRESSION MODEL

```
credit = pd.read_csv('Credit.csv')
credit[:5]
```

Y

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331

```
credit.shape
```

```
(400, 11)
```

(n, p)

BEST REGRESSION MODEL – ENCODE CATEGORICAL VARS

```
credit[:5]
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331

```
credit1 = pd.get_dummies(credit,
                         columns = ['Gender','Student','Married','Ethnicity'],
                         drop_first = True)
credit1[:5]
```

	Income	Limit	Rating	Cards	Age	Education	Balance	Gender_Female	Student_Yes	Married_Yes	Ethnicity_Asian	Ethnicity_Caucasian
0	14.891	3606	283	2	34	11	333	0	0	1	0	1
1	106.025	6645	483	3	82	15	903	1	1	1	1	0
2	104.593	7075	514	4	71	11	580	0	0	0	1	0
3	148.924	9504	681	3	36	11	964	1	0	0	1	0
4	55.882	4897	357	2	68	16	331	0	0	1	0	1

BEST REGRESSION MODEL – ENCODE CATEGORICAL VARS

```
credit[:5]
```

Y

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
2	104.593	7075	514	4	71	11	Male	No	No	Asian	580
3	148.924	9504	681	3	36	11	Female	No	No	Asian	964
4	55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331

```
credit1 = pd.get_dummies(credit,
                         columns = ['Gender','Student','Married','Ethnicity'],
                         drop_first = True)
credit1[:5]
```

Y

	Income	Limit	Rating	Cards	Age	Education	Balance	Gender_Female	Student_Yes	Married_Yes	Ethnicity_Asian	Ethnicity_Caucasian
0	14.891	3606	283	2	34	11	333	0	0	1	0	1
1	106.025	6645	483	3	82	15	903	1	1	1	1	0
2	104.593	7075	514	4	71	11	580	0	0	0	1	0
3	148.924	9504	681	3	36	11	964	1	0	0	1	0
4	55.882	4897	357	2	68	16	331	0	0	1	0	1

BEST REGRESSION MODEL – ENCODE CATEGORICAL VARS

```
credit2 = credit1.drop('Balance',axis=1)
credit2[:5]
```

	Income	Limit	Rating	Cards	Age	Education	Gender_Female	Student_Yes	Married_Yes	Ethnicity_Asian	Ethnicity_Caucasian
0	14.891	3606	283	2	34	11	0	0	1	0	1
1	106.025	6645	483	3	82	15	1	1	1	1	0
2	104.593	7075	514	4	71	11	0	0	0	1	0
3	148.924	9504	681	3	36	11	1	0	0	1	0
4	55.882	4897	357	2	68	16	0	0	1	0	1

```
credit2.shape
```

```
(400, 11)
```

How many models with 2 predictors? →

```
comb(11,2)
```

```
55.0
```

How many models with 3 predictors? →

```
comb(11,3)
```

```
165.0
```

How many models with 0,1,…,11 predictors? →

```
2**11
```

```
2048
```

BEST REGRESSION MODEL – Split data into a train and a test set

```
y0 = credit1.Balance
X0 = credit1.drop(columns = 'Balance', axis = 1)
```

```
X,X_test,y,y_test = train_test_split(X0,y0,
                                     test_size = 0.25,
                                     random_state=1)
```

```
# train sets are X,y
# test sets are X_test,y_test
```

```
print(X.shape,X_test.shape)
```

```
(300, 11) (100, 11)
```

```
# train set size
n = 300
```

Build Model with all predictors (finding its MSPE)

Holdout Cross Validation

```
model0 = LinearRegression().fit(X,y)
```

build model with the train set

```
# MSPE
```

```
yhat0 = model0.predict(X_test)
MSPE = mean_squared_error(y_test,yhat0)
MSPE
```

predict with test set

```
11521.699081990417
```

```
np.sqrt(MSPE)
```

```
107.33917775905691
```

Build Model with all predictors (finding its MSPE)

K-fold Cross Validation (K = 10)

```
kfold = KFold(n_splits=10,random_state=1,shuffle = True)
```

```
mspe = cross_val_score(LinearRegression(),X0,y0,
                       cv = kfold,
                       scoring = 'neg_mean_squared_error')
```

```
mspe
```

```
array([ -9977.31246066, -16049.63211923,  -8256.55274834, -10718.06396806,
        -9983.7567259 , -10455.00758441,  -9675.20399605,  -9789.97445626,
        -6454.62129103, -10230.20348514])
```

```
-mspe.mean()
```

```
10159.032883508195
```

```
np.sqrt(-mspe.mean())
```

```
100.7920278767532
```

Choose the best set of predictors for the regression model using AIC and Adj R-squared

FIND SSE, R-SQUARED – FULL MODEL

A function returning SSE, R-squared

```
def get_sse(X,Y):
    model = LinearRegression().fit(X,Y)
    yhat = model.predict(X)
    SSE = mean_squared_error(Y,yhat)*n
    R_squared = model.score(X,Y)
    return SSE, R_squared
```

FIND SSE, R-SQUARED – FULL MODEL

```
# train sets are X,y
# test sets are X_test,y_test
```

A function returning SSE, R-squared

```
def get_sse(X,Y):
    model = LinearRegression().fit(X,Y)
    yhat = model.predict(X)
    SSE = mean_squared_error(Y,yhat)*n
    R_squared = model.score(X,Y)
    return SSE, R_squared
```

```
SSE, R_squared = get_sse(X,y)
```

```
SSE
```

```
2695832.245813148
```

```
R_squared
```

```
0.9557279779952305
```

FIND SSE, R-SQUARED – ALL MODELS

```
list(X.columns)
```

```
['Income',
 'Limit',
 'Rating',
 'Cards',
 'Age',
 'Education',
 'Gender_Female',
 'Student_Yes',
 'Married_Yes',
 'Ethnicity_Asian',
 'Ethnicity_Caucasian']
```

```
p = len(X.columns)
p
```

```
11
```

```
list(range(1,p+1))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

FIND SSE, R-SQUARED – ALL MODELS

```
list(X.columns)
```

```
['Income',
 'Limit',
 'Rating',
 'Cards',
 'Age',
 'Education',
 'Gender_Female',
 'Student_Yes',
 'Married_Yes',
 'Ethnicity_Asian',
 'Ethnicity_Caucasian']
```

```
p = len(X.columns)
p
```

```
11
```

```
list(range(1,p+1))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
# all sets of two column names
```

```
list(itertools.combinations(X.columns,2))
```

```
[('Income', 'Limit'),
 ('Income', 'Rating'),
 ('Income', 'Cards'),
 ('Income', 'Age'),
 ('Income', 'Education'),
 ('Income', 'Gender_Female'),
 ('Income', 'Student_Yes'),
 ('Income', 'Married_Yes'),
 ('Income', 'Ethnicity_Asian'),
 ('Income', 'Ethnicity_Caucasian'),
 ('Limit', 'Rating'),
 ('Limit', 'Cards'),
 ⋮
```

FIND SSE, R-SQUARED – ALL MODELS

fit all 2047 models getting their SSE, R-squared

```
SSE_list, R2_list, feature_list, num_features = [],[],[],[]
```

a nested loop (to find SSE and R-squared from each model)

```
for k in range(1,p+1):
    for subset in itertools.combinations(X.columns,k):
        feature_list.append(subset)
        num_features.append(len(subset))
```

k is the number of predictors in the model

← select a subset of k predictors from the 11 variables adding their names to the lists

FIND SSE, R-SQUARED – ALL MODELS

fit all 2047 models getting their SSE, R-squared

```
SSE_list, R2_list, feature_list, num_features = [],[],[],[]
```

a nested loop (to find SSE and R-squared from each model)

```
for k in range(1,p+1):
    for subset in itertools.combinations(X.columns,k):
        feature_list.append(subset)
        num_features.append(len(subset))

        X2 = X[list(subset)]
        SSE, R_squared = get_sse(X2,y)

        SSE_list.append(SSE)
        R2_list.append(R_squared)
```

k is the number of predictors in the model

← select a subset of k predictors from the 11 variables adding their names to the lists

← subset dataframe X with the columns of selected predictors

← build the model, get R-squared

add SSE, R2 to the lists

at the end, each list has 2047 entries

BEST REGRESSION MODEL – FIT ALL MODELS

```
num_features[:5]
```

```
[1, 1, 1, 1, 1]
```

```
feature_list[:5]
```

```
[('Income',), ('Limit',), ('Rating',), ('Cards',), ('Age',)]
```

```
R2_list[:5]
```

```
[0.2200181327359455,
 0.7568493163363379,
 0.7616305879459113,
 0.014850886580138112,
 0.0005190442873671541]
```

```
SSE_list[:5]
```

```
[47495013.18673583,
 14806042.821838915,
 14514899.44059262,
 59988151.130874075,
 60860852.23746908]
```

Create a dataframe with 4 columns filled with these 4 lists

	n_features	features	R-squared	SSE
0	1	(Income,)	0.220018	47495013.0
1	1	(Limit,)	0.756849	14806043.0
2	1	(Rating,)	0.761631	14514899.0
3	1	(Cards,)	0.014851	59988151.0
4	1	(Age,)	0.000519	60860852.0

BEST REGRESSION MODEL

```
# create dataframe with the four lists
```

```
zip1 = zip(num_features,feature_list,R2_list,SSE_list)
df = pd.DataFrame(list(zip1),
                  columns = ['n_features','features','R-squared','SSE'])
```

```
df['SSE'] = df['SSE'].round(0)
df[:5]
```

	n_features	features	R-squared	SSE
0	1	(Income,)	0.220018	47495013.0
1	1	(Limit,)	0.756849	14806043.0
2	1	(Rating,)	0.761631	14514899.0
3	1	(Cards,)	0.014851	59988151.0
4	1	(Age,)	0.000519	60860852.0

```
df.shape
```

```
(2047, 4)
```

9 REGRESSION MODELS

.

```
df.sample(9, random_state=1)
```

	n_features	features	R-squared	SSE
309	4	(Income, Cards, Gender_Female, Ethnicity_Cauca...	0.242865	46103833.0
1199	6	(Income, Rating, Age, Gender_Female, Ethnicity...	0.879609	7330923.0
1755	7	(Limit, Cards, Age, Education, Gender_Female, ...	0.771386	13920841.0
563	5	(Income, Limit, Rating, Cards, Gender_Female)	0.878893	7374524.0
56	2	(Gender_Female, Student_Yes)	0.073873	56394170.0
546	4	(Education, Gender_Female, Student_Yes, Marrie...	0.075743	56280289.0
1293	6	(Limit, Rating, Cards, Education, Gender_Femal...	0.765966	14250915.0
181	3	(Cards, Education, Gender_Female)	0.025366	59347877.0
613	5	(Income, Limit, Age, Education, Ethnicity_Asian)	0.872174	7783615.0

→|

9 REGRESSION MODELS

```
pd.set_option('display.max_colwidth',190)
```

```
df.sample(9, random_state=1)
```

	n_features	features	R-squared	SSE
309	4	(Income, Cards, Gender_Female, Ethnicity_Caucasian)	0.242865	46103833.0
1199	6	(Income, Rating, Age, Gender_Female, Ethnicity_Asian, Ethnicity_Caucasian)	0.879609	7330923.0
1755	7	(Limit, Cards, Age, Education, Gender_Female, Married_Yes, Ethnicity_Caucasian)	0.771386	13920841.0
563	5	(Income, Limit, Rating, Cards, Gender_Female)	0.878893	7374524.0
56	2	(Gender_Female, Student_Yes)	0.073873	56394170.0
546	4	(Education, Gender_Female, Student_Yes, Married_Yes)	0.075743	56280289.0
1293	6	(Limit, Rating, Cards, Education, Gender_Female, Ethnicity_Caucasian)	0.765966	14250915.0
181	3	(Cards, Education, Gender_Female)	0.025366	59347877.0
613	5	(Income, Limit, Age, Education, Ethnicity_Asian)	0.872174	7783615.0

CHOOSE BEST MODELS

```
pd.set_option('display.max_colwidth',190)
```

```
df.sample(9, random_state=1)
```

	n_features	features	R-squared	SSE
309	4	(Income, Cards, Gender_Female, Ethnicity_Caucasian)	0.242865	46103833.0
1199	6	(Income, Rating, Age, Gender_Female, Ethnicity_Asian, Ethnicity_Caucasian)	0.879609	7330923.0
1755	7	(Limit, Cards, Age, Education, Gender_Female, Married_Yes, Ethnicity_Caucasian)	0.771386	13920841.0
563	5	(Income, Limit, Rating, Cards, Gender_Female)	0.878893	7374524.0
56	2	(Gender_Female, Student_Yes)	0.073873	56394170.0
546	4	(Education, Gender_Female, Student_Yes, Married_Yes)	0.075743	56280289.0
1293	6	(Limit, Rating, Cards, Education, Gender_Female, Ethnicity_Caucasian)	0.765966	14250915.0
181	3	(Cards, Education, Gender_Female)	0.025366	59347877.0
613	5	(Income, Limit, Age, Education, Ethnicity_Asian)	0.872174	7783615.0

BEST MODELS BY THE NUMBER OF PREDICTORS

```
# Largest R-squared by n_features
```

```
best_r2 = df.groupby(['n_features'])['R-squared'].max()
best_r2
```

```
n_features
1     0.761631
2     0.877077
3     0.951700
4     0.952881
5     0.954163
6     0.955137
7     0.955429
8     0.955688
9     0.955719
10    0.955724
11    0.955728
Name: R-squared, dtype: float64
```

BEST MODELS BY THE NUMBER OF PREDICTORS

```
# Largest R-squared by n_features
```

```
best_r2 = df.groupby(['n_features'])['R-squared'].max()
best_r2
```

```
n_features
1      0.761631
2      0.877077
3      0.951700
4      0.952881
5      0.954163
6      0.955137
7      0.955429
8      0.955688
9      0.955719
10     0.955724
11     0.955728
Name: R-squared, dtype: float64
```

best_r2[1] is the R^2 of the best model with one predictor

BEST REGRESSION MODEL WITH 1 FEATURE

```
# Largest R-squared by n_features
```

```
best_r2 = df.groupby(['n_features'])['R-squared'].max()
best_r2
```

```
n_features
1       0.761631
2       0.877077
3       0.951700
4       0.952881
5       0.954163
6       0.955137
7       0.955429
8       0.955688
9       0.955719
10      0.955724
11      0.955728
Name: R-squared, dtype: float64
```

```
# Best model with one feature
df2 = df[df['R-squared'] == best_r2[1]]
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0

BEST REGRESSION MODEL WITH 2 FEATURES

```
# Largest R-squared by n_features
```

```
best_r2 = df.groupby(['n_features'])['R-squared'].max()
best_r2
```

```
n_features
1      0.761631
2      0.877077
3      0.951700
4      0.952881
5      0.954163
6      0.955137
7      0.955429
8      0.955688
9      0.955719
10     0.955724
11     0.955728
Name: R-squared, dtype: float64
```

```
# Best model with two features
df2 = df[df['R-squared'] == best_r2[2]]
df2
```

	n_features	features	R-squared	SSE
12	2	(Income, Rating)	0.877077	7485072.0

BEST REGRESSION MODEL WITH 2 FEATURES

```
# Best model with one feature
df2 = df[df['R-squared'] == best_r2[1]]
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0

```
# Best model with two features
df2 = df[df['R-squared'] == best_r2[2]]
df2
```

	n_features	features	R-squared	SSE
12	2	(Income, Rating)	0.877077	7485072.0

BEST REGRESSION MODEL WITH 2 FEATURES

```
# Best model with one feature
df2 = df[df['R-squared'] == best_r2[1]]
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0

```
# Best model with two features
df2 = df[df['R-squared'] == best_r2[2]]
df2
```

	n_features	features	R-squared	SSE
12	2	(Income, Rating)	0.877077	7485072.0

```
for i in range(1,12):
    df2 = df[df['R-squared'] == best_r2[i]]
    print(df2.index.values)
```

```
[2]
[12]
[79]
[235]
[564]
[1025]
[1495]
[1833]
[1988]
[2039]
[2046]
```

BEST REGRESSION MODEL WITH 2 FEATURES

```
# Best model with one feature
df2 = df[df['R-squared'] == best_r2[1]]
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0

```
# Best model with two features
df2 = df[df['R-squared'] == best_r2[2]]
df2
```

	n_features	features	R-squared	SSE
12	2	(Income, Rating)	0.877077	7485072.0

```
for i in range(1,12):
    df2 = df[df['R-squared'] == best_r2[i]]
    print(df2.index.values)
```

```
[2]
[12]
[79]
[235]
[564]
[1025]
[1495]
[1833]
[1988]
[2039]
[2046]
```

```
list1 = list()
for i in range(1,12):
    df2 = df[df['R-squared'] == best_r2[i]]
    list1.extend(df2.index.values.tolist())
```

```
list1
```

```
[2, 12, 79, 235, 564, 1025, 1495, 1833, 1988,
```

DATAFRAME WITH BEST 11 MODELS

```python
df2 = df.iloc[list1].copy()
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0
12	2	(Income, Rating)	0.877077	7485072.0
79	3	(Income, Rating, Student_Yes)	0.951700	2941135.0
235	4	(Income, Limit, Rating, Student_Yes)	0.952881	2869207.0
564	5	(Income, Limit, Rating, Cards, Student_Yes)	0.954163	2791149.0
1025	6	(Income, Limit, Rating, Cards, Age, Student_Yes)	0.955137	2731792.0
1495	7	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian)	0.955429	2714033.0
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955688	2698241.0
1988	9	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955719	2696376.0
2039	10	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955724	2696091.0
2046	11	(Income, Limit, Rating, Cards, Age, Education, Gender_Female, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955728	2695832.0

DATAFRAME WITH BEST 11 MODELS

```
df2 = df.iloc[list1].copy()
df2
# best predictor is Rating,
# Worst predictor is Gender
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0
12	2	(Income, Rating)	0.877077	7485072.0
79	3	(Income, Rating, Student_Yes)	0.951700	2941135.0
235	4	(Income, Limit, Rating, Student_Yes)	0.952881	2869207.0
564	5	(Income, Limit, Rating, Cards, Student_Yes)	0.954163	2791149.0
1025	6	(Income, Limit, Rating, Cards, Age, Student_Yes)	0.955137	2731792.0
1495	7	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian)	0.955429	2714033.0
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955688	2698241.0
1988	9	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955719	2696376.0
2039	10	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955724	2696091.0
2046	11	(Income, Limit, Rating, Cards, Age, Education, Gender_Female, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955728	2695832.0

DATAFRAME WITH BEST 11 MODELS

Cannot compare these models using R-squared

```
df2 = df.iloc[list1].copy()
df2
```

	n_features	features	R-squared	SSE
2	1	(Rating,)	0.761631	14514899.0
12	2	(Income, Rating)	0.877077	7485072.0
79	3	(Income, Rating, Student_Yes)	0.951700	2941135.0
235	4	(Income, Limit, Rating, Student_Yes)	0.952881	2869207.0
564	5	(Income, Limit, Rating, Cards, Student_Yes)	0.954163	2791149.0
1025	6	(Income, Limit, Rating, Cards, Age, Student_Yes)	0.955137	2731792.0
1495	7	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian)	0.955429	2714033.0
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955688	2698241.0
1988	9	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955719	2696376.0
2039	10	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955724	2696091.0
2046	11	(Income, Limit, Rating, Cards, Age, Education, Gender_Female, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.955728	2695832.0

ADJUSTER R-SQUARED FORMULA

.

$$adj\ R^2 = 1 - \frac{MSE}{MST}$$

$$= 1 - \frac{\frac{SSE}{n-p-1}}{\frac{SST}{n-1}}$$

$$adj\ R^2 = 1 - \frac{n-1}{n-p-1}\frac{SSE}{SST}$$

ADJUSTER R-SQUARED FORMULA

.

$$1 - adj\ R^2 = \frac{n-1}{n-p-1} \frac{SSE}{SST}$$

$$1 - adj\ R^2 = \frac{n-1}{n-p-1}(1 - R^2)$$

$$adj\ R^2 = 1 - \frac{n-1}{n-p-1}(1 - R^2)$$

ADJUSTER R-SQUARED AND AIC FORMULAS

.

$$adj\ R^2 = 1 - \frac{n-1}{n-p-1}(1-R^2)$$

$$AIC = n \log\left(\frac{SSE}{n}\right) + 2p$$

BEST REGRESSION MODEL

.

```
# Add columns for AIC, adj R-squared
```

```
df2['adj_R-squared'] = 1 - ((1-df2['R-squared'])*(n-1)/
                            (n - df2['n_features'] - 1))
df2['AIC'] = n * np.log(df2['SSE']/n) + 2*df2['n_features']
```

```
# Remove R-squared, SSE columns
```

```
df2.drop(['R-squared','SSE'],axis=1,inplace=True)
```

DATAFRAME WITH BEST 11 MODELS

```
df2
```

	n_features	features	adj_R-squared	AIC
2	1	(Rating,)	0.760831	3238.071117
12	2	(Income, Rating)	0.876249	3041.391616
79	3	(Income, Rating, Student_Yes)	0.951210	2763.157093
235	4	(Income, Limit, Rating, Student_Yes)	0.952242	2757.729130
564	5	(Income, Limit, Rating, Cards, Student_Yes)	0.953383	2751.454427
1025	6	(Income, Limit, Rating, Cards, Age, Student_Yes)	0.954219	2747.005766
1495	7	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian)	0.954361	2747.049141
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954470	2747.298449
1988	9	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954345	2749.091020
2039	10	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954192	2751.059309
2046	11	(Income, Limit, Rating, Cards, Age, Education, Gender_Female, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954037	2753.030488

BEST MODELS

```
df2
```

	n_features	features	adj_R-squared	AIC
2	1	(Rating,)	0.760831	3238.071117
12	2	(Income, Rating)	0.876249	3041.391616
79	3	(Income, Rating, Student_Yes)	0.951210	2763.157093
235	4	(Income, Limit, Rating, Student_Yes)	0.952242	2757.729130
564	5	(Income, Limit, Rating, Cards, Student_Yes)	0.953383	2751.454427
1025	6	best AIC model (Income, Limit, Rating, Cards, Age, Student_Yes)	0.954219	2747.005766
1495	7	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian)	0.954361	2747.049141
1833	8	best Adj-R^2 model (Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954470	2747.298449
1988	9	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954345	2749.091020
2039	10	(Income, Limit, Rating, Cards, Age, Education, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954192	2751.059309
2046	11	(Income, Limit, Rating, Cards, Age, Education, Gender_Female, Student_Yes, Married_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.954037	2753.030488

AIC Best Model

AIC BEST MODEL

```
# AIC best model has 6 features
```

```
row6 = df2[df2.AIC == df2.AIC.min()]
row6
```

	n_features	features	adj_R-squared	AIC
1025	6	(Income, Limit, Rating, Cards, Age, Student_Yes)	0.954219	2747.005766

```
# get predictor names of AIC best model
```

```
row6.features.iloc[0]
```

```
('Income', 'Limit', 'Rating', 'Cards', 'Age', 'Student_Yes')
```

AIC BEST MODEL – Create DataFrame X1 with columns for AIC best model

```
list1 = list(row6.features.iloc[0])
list1
```

```
['Income', 'Limit', 'Rating', 'Cards', 'Age', 'Student_Yes']
```

```
# make dataset with these columns only
```

```
X1 = X[list1]
X1[:5]
```

	Income	Limit	Rating	Cards	Age	Student_Yes
82	23.672	4433	344	3	63	0
367	23.793	3615	263	2	70	0
179	58.026	7499	560	5	67	0
27	32.793	4534	333	2	44	0
89	59.530	7518	543	3	52	0

Adj. R-squared Best Model

adj-R2 BEST MODEL

```
max_R2 = df2['adj_R-squared'].max()
max_R2
```

```
0.9544702381591676
```

```
df2[df2['adj_R-squared'] == max_R2]
```

	n_features	features	adj_R-squared
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.95447

```
# Adj-R2 best model has 8 features
```

adj-R2 BEST MODEL

```
row8 = df2[df2['adj_R-squared'] == max_R2]
row8
```

	n_features	features	adj_R-squared
1833	8	(Income, Limit, Rating, Cards, Age, Student_Yes, Ethnicity_Asian, Ethnicity_Caucasian)	0.95447

```
# get feature names of adj-R2 best model
```

```
row8.features.iloc[0]
```

```
('Income',
 'Limit',
 'Rating',
 'Cards',
 'Age',
 'Student_Yes',
 'Ethnicity_Asian',
 'Ethnicity_Caucasian')
```

adj-R2 BEST MODEL - Create DataFrame X2 with columns for Adj-R2 best model

```
# make dataset X2 with these columns only
```

```
list2 = list(row8.features.iloc[0])
X2 = X[list2]
```

```
X2[:5]
```

	Income	Limit	Rating	Cards	Age	Student_Yes	Ethnicity_Asian	Ethnicity_Caucasian
82	23.672	4433	344	3	63	0	0	1
367	23.793	3615	263	2	70	0	0	0
179	58.026	7499	560	5	67	0	0	1
27	32.793	4534	333	2	44	0	0	0
89	59.530	7518	543	3	52	0	0	0

Use Holdout Cross-validation to compare Best Models

Which is best for prediction?

COMPARE MODELS – FULL Model

Validation Approach

```
# MSPE of full (all predictors) model
```

```
model0 = LinearRegression().fit(X,y)
yhat0 = model0.predict(X_test)
```

```
MSPE = mean_squared_error(y_test,yhat0)
np.sqrt(MSPE)
```

```
107.33917775905691
```

COMPARE MODELS

.

Best Adj R-squared Model

```
model2 = LinearRegression().fit(X2,y)
```

```
X2_test = X_test[list2]
yhat2 = model2.predict(X2_test)
```

```
MSPE = mean_squared_error(y_test,yhat2)
np.sqrt(MSPE)
```

```
107.37550073954571
```

COMPARE MODELS

.

Best AIC Model

```
model1 = LinearRegression().fit(X1,y)
```

```
X1_test = X_test[list1]
yhat1 = model1.predict(X1_test)
```

```
MSPE = mean_squared_error(y_test,yhat1)
np.sqrt(MSPE)
```

```
105.78913727639237
```

```
# AIC model is best predictive model
```

Use Kfold Cross-validation to compare Best Models

Which is best for prediction?

COMPARE MODELS

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

full model -dataset X0

```
kfold = KFold(n_splits=10,random_state=1,shuffle = True)
```

```
# Use all rows of data (Data sets X0,y0)
```

```
mspe = cross_val_score(LinearRegression(),X0,y0,
                       cv = kfold,
                       scoring = 'neg_mean_squared_error')
```

```
-mspe.mean(),
```

```
(10159.032883508191,)
```

```
np.sqrt(-mspe.mean())
```

```
100.79202787675318
```

COMPARE MODELS

best adj-R2 model -dataset X2

```
X2 = X0[list2]
```

```
mspe = cross_val_score(LinearRegression(),X2,y0,
                       cv = kfold,
                       scoring = 'neg_mean_squared_error')
```

```
-mspe.mean()
```

```
10102.635473660017
```

```
np.sqrt(-mspe.mean())
```

```
100.5118673274953
```

COMPARE MODELS

best AIC model -dataset X1

```
X1 = X0[list1]
```

```
mspe = cross_val_score(LinearRegression(),X1,y0,
                       cv = kfold,
                       scoring = 'neg_mean_squared_error')
```

```
-mspe.mean()
```

```
10066.384682386239
```

```
np.sqrt(-mspe.mean())
```

```
100.33137436707543
```

```
# AIC model is best predictive model
```

Prediction

PREDICTION WITH BEST AIC MODEL

- Predict the Balance of a credit card customer with median values for income, credit limit, credit rating, number of credit cards, and age
- Assume that the student status of the customer is the most frequent category

PREDICTION WITH BEST AIC MODEL

```
newval = X1.head(1).copy()
newval
```

	Income	Limit	Rating	Cards	Age	Student_Yes
0	14.891	3606	283	2	34	0

```
newval.Income = np.median(X1.Income)
newval.Limit = np.median(X1.Limit)
newval.Rating = np.median(X1.Rating)
newval.Cards = np.median(X1.Cards)
newval.Age = np.median(X1.Age)
```

```
# most common student status
```

```
pd.value_counts(X1.Student_Yes)
```

```
0     360        ← not student
1      40        ← student
Name: Student_Yes, dtype: int64
```

Most common category is non-student

PREDICTION WITH BEST AIC MODEL

```
newval = X1.head(1).copy()
newval
```

	Income	Limit	Rating	Cards	Age	Student_Yes
0	14.891	3606	283	2	34	0

```
newval.Income = np.median(X1.Income)
newval.Limit = np.median(X1.Limit)
newval.Rating = np.median(X1.Rating)
newval.Cards = np.median(X1.Cards)
newval.Age = np.median(X1.Age)
```

```
# most common student status
```

```
pd.value_counts(X1.Student_Yes)
```

```
0    360
1     40
Name: Student_Yes, dtype: int64
```

Most common category is non-student

```
newval
```

predict Balance of this new customer

	Income	Limit	Rating	Cards	Age	Student_Yes
0	33.1155	4622.5	344.0	3.0	56.0	0

PREDICTION

```
# fit model and predict Balance
```

```
model = LinearRegression().fit(X1,y0)
model.predict(newval)
```

```
array([538.52330854])
```

Predicted Customer Balance is 538.52 dollars