

```
In [87]: import pandas #ipython notebook
titanic = pandas.read_csv('/kaggle/input/titanic/train.csv')
titanic.head(5)
```

```
Out[87]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2834
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

```
In [88]: titanic["Age"] = titanic["Age"].fillna(titanic["Age"].median())
print (titanic.describe())
```

	PassengerId	Survived	Pclass	Age	SibSp	\
count	891.000000	891.000000	891.000000	891.000000	891.000000	
mean	446.000000	0.383838	2.308642	29.361582	0.523008	
std	257.353842	0.486592	0.836071	13.019697	1.102743	
min	1.000000	0.000000	1.000000	0.420000	0.000000	
25%	223.500000	0.000000	2.000000	22.000000	0.000000	
50%	446.000000	0.000000	3.000000	28.000000	0.000000	
75%	668.500000	1.000000	3.000000	35.000000	1.000000	
max	891.000000	1.000000	3.000000	80.000000	8.000000	

	Parch	Fare
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200
75%	0.000000	31.000000
max	6.000000	512.329200

```
In [89]: print (titanic["Sex"].unique())

# Replace all the occurrences of male with the number 0.
titanic.loc[titanic["Sex"] == "male", "Sex"] = 0
titanic.loc[titanic["Sex"] == "female", "Sex"] = 1

['male' 'female']
```

```
In [90]: print (titanic["Embarked"].unique())
titanic["Embarked"] = titanic["Embarked"].fillna('S')
titanic.loc[titanic["Embarked"] == "S", "Embarked"] = 0
titanic.loc[titanic["Embarked"] == "C", "Embarked"] = 1
titanic.loc[titanic["Embarked"] == "Q", "Embarked"] = 2

['S' 'C' 'Q' nan]
```

```
In [91]: # Import the linear regression class
from sklearn.linear_model import LinearRegression
# Sklearn also has a helper that makes it easy to do cross validation
from sklearn.model_selection import KFold

# The columns we'll use to predict the target
predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]

# Initialize our algorithm class
alg = LinearRegression()
# Generate cross validation folds for the titanic dataset. It return the row indic
# We set random_state to ensure we get the same splits every time we run this.

kf=KFold( n_splits=3,shuffle=True, random_state=1)
#kf.get_n_splits(titanic.shape[0])
predictions = []
for train, test in kf.split(titanic):
    # The predictors we're using to train the algorithm. Note how we only take the
    train_predictors = (titanic[predictors].iloc[train,:])
    # The target we're using to train the algorithm.
    train_target = titanic["Survived"].iloc[train]
    # Training the algorithm using the predictors and target.
    alg.fit(train_predictors, train_target)
    # We can now make predictions on the test fold
    test_predictions = alg.predict(titanic[predictors].iloc[test,:])
    predictions.append(test_predictions)
```

[0 2 3 6 8 13 14 16 17 19 23 34 35 40 41 46 47 49
50 56 57 58 59 60 61 65 66 68 69 73 74 76 81 84 85 89
90 92 94 101 104 106 107 108 111 114 117 118 119 120 121 125 132 134
135 139 142 148 154 159 160 161 175 180 181 185 187 189 194 195 201 202
205 207 216 218 223 224 228 233 238 239 241 242 244 245 246 247 248 255
257 258 262 265 267 268 274 277 286 289 291 294 298 301 307 309 311 314
318 320 329 331 339 340 345 349 350 354 355 363 364 368 370 371 372 382
383 385 386 404 408 411 412 414 415 421 422 426 427 430 433 435 437 439
443 446 448 449 452 453 463 464 467 473 479 484 487 493 496 498 504 506
507 509 510 512 516 518 521 522 525 528 529 535 543 550 551 554 558 563
565 570 571 573 574 576 578 584 585 589 593 594 597 598 599 600 605 610
613 614 619 622 623 625 629 634 637 640 643 644 649 654 658 659 662 671
674 676 677 678 680 681 684 686 688 692 702 703 708 710 713 719 724 726
729 730 731 733 736 737 739 740 741 742 743 744 748 757 761 765 766 769
772 773 775 789 790 791 794 795 796 797 800 802 803 811 813 815 817 819
821 822 824 827 828 830 832 841 845 848 850 852 859 862 865 866 868 871
874 875 876 877 878 882 885 887 888]
[1 4 5 9 11 12 18 27 29 30 31 33 36 38 39 42 45 52
54 62 67 70 78 80 82 83 88 91 93 95 97 98 99 102 103 105
110 112 115 116 122 124 127 128 133 146 147 150 153 156 157 158 162 164
165 168 171 172 173 177 179 184 186 191 192 197 200 203 204 213 214 217
220 221 225 226 227 232 236 237 249 250 259 260 261 272 273 280 283 284
285 292 293 295 299 304 305 306 308 310 312 315 323 328 333 334 335 337
341 343 344 346 347 351 352 353 358 359 360 361 362 365 366 373 374 375
376 377 378 379 388 389 392 394 395 396 397 399 400 402 403 406 410 417
419 423 425 428 429 432 434 436 438 441 442 445 447 455 457 459 460 472
474 480 481 482 483 486 488 491 494 495 501 502 511 517 519 523 524 527
530 531 537 538 539 540 541 544 546 549 552 553 555 556 559 560 567 577
579 581 582 587 590 592 601 602 603 604 607 608 612 618 624 632 635 636
638 639 641 647 650 651 652 657 661 663 664 666 667 669 670 672 673 679
682 683 685 690 697 698 699 700 701 704 705 709 711 716 718 720 721 723
725 727 735 745 746 747 750 755 756 759 763 768 770 776 779 780 781 782
786 788 805 808 809 810 814 816 818 825 831 837 839 842 844 851 853 855
856 857 858 864 870 873 883 884 890]
[7 10 15 20 21 22 24 25 26 28 32 37 43 44 48 51 53 55
63 64 71 72 75 77 79 86 87 96 100 109 113 123 126 129 130 131
136 137 138 140 141 143 144 145 149 151 152 155 163 166 167 169 170 174
176 178 182 183 188 190 193 196 198 199 206 208 209 210 211 212 215 219
222 229 230 231 234 235 240 243 251 252 253 254 256 263 264 266 269 270
271 275 276 278 279 281 282 287 288 290 296 297 300 302 303 313 316 317
319 321 322 324 325 326 327 330 332 336 338 342 348 356 357 367 369 380
381 384 387 390 391 393 398 401 405 407 409 413 416 418 420 424 431 440
444 450 451 454 456 458 461 462 465 466 468 469 470 471 475 476 477 478
485 489 490 492 497 499 500 503 505 508 513 514 515 520 526 532 533 534
536 542 545 547 548 557 561 562 564 566 568 569 572 575 580 583 586 588
591 595 596 606 609 611 615 616 617 620 621 626 627 628 630 631 633 642
645 646 648 653 655 656 660 665 668 675 687 689 691 693 694 695 696 706
707 712 714 715 717 722 728 732 734 738 749 751 752 753 754 758 760 762
764 767 771 774 777 778 783 784 785 787 792 793 798 799 801 804 806 807
812 820 823 826 829 833 834 835 836 838 840 843 846 847 849 854 860 861
863 867 869 872 879 880 881 886 889]

```
In [92]: import numpy as np

# The predictions are in three separate numpy arrays. Concatenate them into one.
# We concatenate them on axis 0, as they only have one axis.
predictions = np.concatenate(predictions, axis=0)

# Map predictions to outcomes (only possible outcomes are 1 and 0)
predictions[predictions > .5] = 1
predictions[predictions <=.5] = 0
accuracy = sum(predictions[predictions == titanic["Survived"]]) / len(predictions)
print(accuracy)
```

0.13468013468013468

```
In [9]: from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
# Initialize our algorithm
alg = LogisticRegression(random_state=1)
# Compute the accuracy score for all the cross validation folds. (much simpler than
scores = model_selection.cross_val_score(alg, titanic[predictors], titanic["Survived"], cv=5)
# Take the mean of the scores (because we have one for each fold)
print(scores.mean())
```

0.7957351290684623

/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```

In [94]: titanic_test = pandas.read_csv("/kaggle/input/titanic/test.csv")
titanic_test["Age"] = titanic_test["Age"].fillna(titanic["Age"].median())
titanic_test["Fare"] = titanic_test["Fare"].fillna(titanic_test["Fare"].median())
titanic_test.loc[titanic_test["Sex"] == "male", "Sex"] = 0
titanic_test.loc[titanic_test["Sex"] == "female", "Sex"] = 1
titanic_test["Embarked"] = titanic_test["Embarked"].fillna("S")

titanic_test.loc[titanic_test["Embarked"] == "S", "Embarked"] = 0
titanic_test.loc[titanic_test["Embarked"] == "C", "Embarked"] = 1
titanic_test.loc[titanic_test["Embarked"] == "Q", "Embarked"] = 2
predictions=[]
test_predictions = alg.predict(titanic_test[predictors])
predictions.append(test_predictions)

predictions = np.concatenate(predictions, axis=0)
predictions[predictions > .5] = 1
predictions[predictions <=.5] = 0
data={}
for i in range(predictions.size):
    if "PassengerId" in data.keys():
        data["PassengerId"].append(i+892)
    else:
        data["PassengerId"] = [i+892]
    if "Survived" in data.keys():
        data["Survived"].append(predictions[i])
    else:
        data["Survived"] = [predictions[i]]
df = pandas.DataFrame(data)

# 导出到 CSV 文件
df.to_csv('/kaggle/working/submission.csv', index=False)

```

```

In [12]: from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier

predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]

# Initialize our algorithm with the default paramters
# n_estimators is the number of trees we want to make
# min_samples_split is the minimum number of rows we need to make a split
# min_samples_leaf is the minimum number of samples we can have at the place where
alg = RandomForestClassifier(random_state=1, n_estimators=10, min_samples_split=2,
# Compute the accuracy score for all the cross validation folds. (much simpler than
kf=KFold( n_splits=3, shuffle=True, random_state=1)
scores = model_selection.cross_val_score(alg, titanic[predictors], titanic["Survived"],
# Take the mean of the scores (because we have one for each fold)
print(scores.mean())

```

0.7957351290684626

```
In [13]: alg = RandomForestClassifier(random_state=1, n_estimators=100, min_samples_split=4,
# Compute the accuracy score for all the cross validation folds. (much simpler than
kf=KFold( 3, shuffle=True, random_state=1)
scores = model_selection.cross_val_score(alg, titanic[predictors], titanic["Survived"],
# Take the mean of the scores (because we have one for each fold)
print(scores.mean()))
```

0.8215488215488215

```
In [15]: # Generating a familysize column
titanic["FamilySize"] = titanic["SibSp"] + titanic["Parch"]

# The .apply method generates a new series
titanic["NameLength"] = titanic["Name"].apply(lambda x: len(x))
```



```
In [16]: import re

# A function to get the title from a name.
def get_title(name):
    # Use a regular expression to search for a title. Titles always consist of capital letters and a period.
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""

# Get all the titles and print how often each one occurs.
titles = titanic["Name"].apply(get_title)
print(pandas.value_counts(titles))

# Map each title to an integer. Some titles are very rare, and are compressed into the same integer.
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Dr": 5, "Rev": 6, "Major": 7, "Countess": 8, "Capt": 9, "Ms": 10, "Sir": 11, "Lady": 12, "Mme": 13, "Don": 14, "Jonkheer": 15}
for k,v in title_mapping.items():
    titles[titles == k] = v

# Verify that we converted everything.
print(pandas.value_counts(titles))

# Add in the title column.
titanic["Title"] = titles
```

```
Name
Mr      517
Miss    182
Mrs     125
Master   40
Dr        7
Rev        6
Mlle        2
Major        2
Col          2
Countess     1
Capt        1
Ms           1
Sir           1
Lady          1
Mme           1
Don           1
Jonkheer      1
Name: count, dtype: int64

Name
1      517
2      183
3      125
4       40
5        7
6         6
7         5
8         3
10        3
9         2
Name: count, dtype: int64
```

```
/tmp/ipykernel_33/2428372315.py:14: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd.Series(obj).value_counts() instead.
```

```
    print(pandas.value_counts(titles))
```

```
/tmp/ipykernel_33/2428372315.py:22: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd.Series(obj).value_counts() instead.
```

```
    print(pandas.value_counts(titles))
```

```
In [17]: import numpy as np
from sklearn.feature_selection import SelectKBest, f_classif
import matplotlib.pyplot as plt
predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "Family

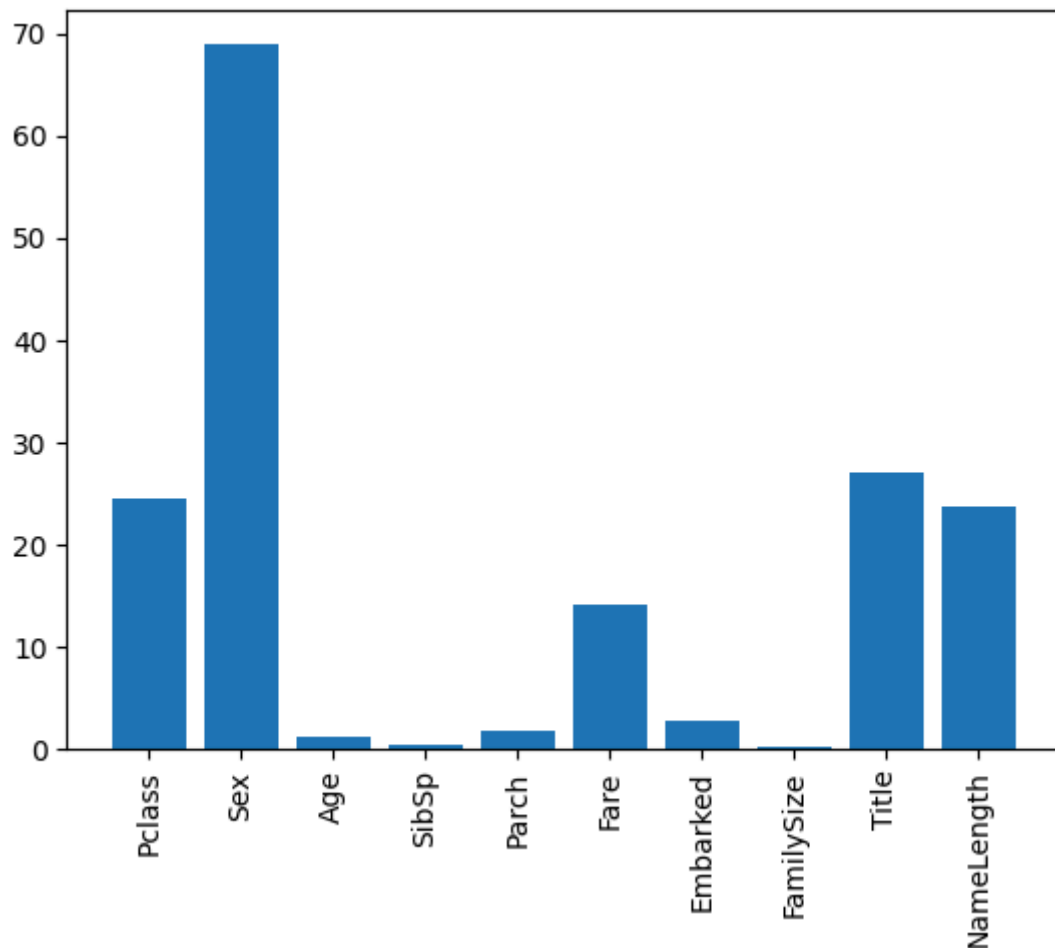
# Perform feature selection
selector = SelectKBest(f_classif, k=5)
selector.fit(titanic[predictors], titanic["Survived"])

# Get the raw p-values for each feature, and transform from p-values into scores
scores = -np.log10(selector.pvalues_)

# Plot the scores. See how "Pclass", "Sex", "Title", and "Fare" are the best?
plt.bar(range(len(predictors)), scores)
plt.xticks(range(len(predictors)), predictors, rotation='vertical')
plt.show()

# Pick only the four best features.
predictors = ["Pclass", "Sex", "Fare", "Title"]

alg = RandomForestClassifier(random_state=1, n_estimators=50, min_samples_split=8,
```



```

In [18]: from sklearn.ensemble import GradientBoostingClassifier
import numpy as np

# The algorithms we want to ensemble.
# We're using the more linear predictors for the logistic regression, and everything
algorithms = [
    GradientBoostingClassifier(random_state=1, n_estimators=25, max_depth=3), ["Pclass", "Sex", "Fare", "FamilySize", "Ticket"]
]

# Initialize the cross validation folds
kf=KFold( n_splits=3, shuffle=True, random_state=1)

predictions = []
for train, test in kf.split(titanic):
    train_target = titanic["Survived"].iloc[train]
    full_test_predictions = []
    # Make predictions for each algorithm on each fold
    for alg, predictors in algorithms:
        # Fit the algorithm on the training data.
        alg.fit(titanic[predictors].iloc[train,:], train_target)
        # Select and predict on the test fold.
        # The .astype(float) is necessary to convert the dataframe to all floats and avoid any integer overflow issues
        test_predictions = alg.predict_proba(titanic[predictors].iloc[test,:].astype(float))[:,1]
        full_test_predictions.append(test_predictions)
    # Use a simple ensembling scheme -- just average the predictions to get the final prediction
    test_predictions = (full_test_predictions[0] + full_test_predictions[1]) / 2
    # Any value over .5 is assumed to be a 1 prediction, and below .5 is a 0 prediction
    test_predictions[test_predictions <= .5] = 0
    test_predictions[test_predictions > .5] = 1
    predictions.append(test_predictions)

# Put all the predictions together into one array.
predictions = np.concatenate(predictions, axis=0)

# Compute accuracy by comparing to the training data.
accuracy = sum(predictions[predictions == titanic["Survived"]]) / len(predictions)
print(accuracy)

```

0.13916947250280584

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
In [19]: titles = titanic_test["Name"].apply(get_title)
# We're adding the Dona title to the mapping, because it's in the test set, but not
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Dr": 5, "Rev": 6, "Major": 7, "Dona": 8}
for k,v in title_mapping.items():
    titles[titles == k] = v
titanic_test["Title"] = titles
# Check the counts of each unique title.
print(pandas.value_counts(titanic_test["Title"]))

# Now, we add the family size column.
titanic_test["FamilySize"] = titanic_test["SibSp"] + titanic_test["Parch"]
```

Title

1	240
2	79
3	72
4	21
7	2
6	2
5	1
10	1

Name: count, dtype: int64

```
/tmp/ipykernel_33/4141970794.py:8: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd.Series(obj).value_counts() instead.
```

```
print(pandas.value_counts(titanic_test["Title"]))
```

```
In [20]: predictors = ["Pclass", "Sex", "Age", "Fare", "Embarked", "FamilySize", "Title"]

algorithms = [
    GradientBoostingClassifier(random_state=1, n_estimators=25, max_depth=3), pred
    LogisticRegression(random_state=1), ["Pclass", "Sex", "Fare", "FamilySize", "T
]

full_predictions = []
for alg, predictors in algorithms:
    # Fit the algorithm using the full training data.
    alg.fit(titanic[predictors], titanic["Survived"])
    # Predict using the test dataset. We have to convert all the columns to floats
    predictions = alg.predict_proba(titanic_test[predictors].astype(float))[:,1]
    full_predictions.append(predictions)

# The gradient boosting classifier generates better predictions, so we weight it hi
predictions = (full_predictions[0] * 3 + full_predictions[1]) / 4
predictions
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: C
onvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
Out[20]: array([0.11608616, 0.47313163, 0.12527406, 0.1305029 , 0.52004311,
0.14392415, 0.63984341, 0.18138299, 0.6778233 , 0.12095368,
0.12054308, 0.21154493, 0.91148439, 0.10814111, 0.89180609,
0.87818912, 0.16501959, 0.13910405, 0.53990168, 0.551889 ,
0.22474624, 0.53695164, 0.90686439, 0.39533671, 0.88099058,
0.10288684, 0.9105501 , 0.13737153, 0.31376952, 0.12631623,
0.11615631, 0.18430576, 0.54919197, 0.49415142, 0.42910391,
0.14215743, 0.50825223, 0.52417057, 0.13230706, 0.28222509,
0.11076129, 0.47224517, 0.09914069, 0.83492474, 0.90028243,
0.14946376, 0.31905677, 0.1375202 , 0.89021873, 0.53830769,
0.36218205, 0.17945766, 0.83411368, 0.87864985, 0.17768223,
0.13743948, 0.10594413, 0.12307288, 0.12054944, 0.912131 ,
0.13115382, 0.15454243, 0.13002401, 0.66586834, 0.66161484,
0.87338585, 0.67298304, 0.29010609, 0.35826283, 0.84875193,
0.66211579, 0.12688447, 0.55224753, 0.37379732, 0.91064833,
0.40981952, 0.12962877, 0.83764819, 0.15754935, 0.66211579,
0.6815184 , 0.199957 , 0.20583641, 0.12054308, 0.18761829,
0.13090738, 0.65624963, 0.53041797, 0.6541947 , 0.80328565,
0.5364396 , 0.12053355, 0.89344775, 0.12962877, 0.29113646,
0.12308551, 0.86391073, 0.14611431, 0.58582725, 0.12192637,
0.90507833, 0.14872081, 0.1375202 , 0.1222069 , 0.62296229,
0.13079696, 0.14629089, 0.1375202 , 0.12968111, 0.17805572,
0.14320048, 0.65420051, 0.89686624, 0.67202898, 0.87958304,
0.14013126, 0.11761531, 0.69866897, 0.369716 , 0.86312987,
0.87861506, 0.12587932, 0.90367706, 0.12049219, 0.1375202 ,
0.56989534, 0.12588394, 0.63632567, 0.13347646, 0.13308618,
0.12658342, 0.51402224, 0.23707771, 0.10764469, 0.09812515,
0.12399516, 0.13310033, 0.16414568, 0.51965018, 0.12217545,
0.20839696, 0.905255 , 0.19234654, 0.16288954, 0.43252004,
0.10460664, 0.34145835, 0.1349816 , 0.47224517, 0.34281095,
0.9150228 , 0.13168701, 0.10605776, 0.48641364, 0.11240039,
0.12396799, 0.91030126, 0.57927986, 0.43252004, 0.51097235,
0.65419138, 0.57892305, 0.82284028, 0.12047252, 0.28604791,
0.58375962, 0.30315379, 0.14606114, 0.90355986, 0.52205389,
0.12051582, 0.13260088, 0.12396504, 0.13162456, 0.13180211,
0.87544578, 0.87770527, 0.29748826, 0.83379914, 0.85394535,
0.15754935, 0.33614481, 0.90334825, 0.1375202 , 0.91674517,
0.13621589, 0.85707224, 0.12265744, 0.14145929, 0.13585733,
0.13547124, 0.2615505 , 0.49879649, 0.12640651, 0.72288221,
0.1072762 , 0.85708385, 0.5900343 , 0.16902875, 0.53774638,
0.64770495, 0.66407876, 0.60484309, 0.87648409, 0.16545898,
0.26353344, 0.62898005, 0.16722597, 0.89151239, 0.12309417,
0.127611 , 0.12047645, 0.24831086, 0.79802557, 0.41149964,
0.30039786, 0.65422041, 0.21582703, 0.89844275, 0.12962877,
0.81556725, 0.13597309, 0.84397565, 0.12687469, 0.87848248,
0.59630913, 0.12492255, 0.6541947 , 0.11396013, 0.14478507,
0.25316202, 0.89445045, 0.11621755, 0.13753447, 0.34495661,
0.12790127, 0.19280511, 0.1403001 , 0.81278248, 0.8975778 ,
0.8764603 , 0.8246372 , 0.32985936, 0.12054243, 0.33097517,
0.28960309, 0.88001689, 0.16051699, 0.86312987, 0.58951613,
0.74977708, 0.15427547, 0.39857128, 0.13320041, 0.12632896,
0.12051582, 0.1375202 , 0.12962877, 0.83210887, 0.12687265,
0.10829711, 0.1268804 , 0.85114827, 0.65202653, 0.16804208,
0.12054308, 0.22527115, 0.12051582, 0.50825223, 0.14031173,
0.34621912, 0.1375202 , 0.91585286, 0.63242281, 0.13162416,
0.85927352, 0.16037479, 0.12507105, 0.14380903, 0.17086699,
0.51978253, 0.66306258, 0.6541947 , 0.64321196, 0.71428972,
0.10527009, 0.12049219, 0.36739267, 0.13162456, 0.12962877,
0.33858104, 0.59221077, 0.13162456, 0.50262011, 0.120044 ,
0.12221704, 0.78214546, 0.12631623, 0.33515934, 0.11974185,
0.11749083, 0.17758583, 0.12164696, 0.133141 , 0.6541947 ,
```

0.8211081 , 0.33592591, 0.67771417, 0.20869482, 0.42050239,
0.13919512, 0.13794091, 0.12051779, 0.61683763, 0.89999581,
0.67464103, 0.23647358, 0.1765905 , 0.1213635 , 0.18705969,
0.1222069 , 0.13464434, 0.16414568, 0.46045705, 0.90515213,
0.12485393, 0.86892053, 0.34709 , 0.14585039, 0.17320123,
0.81934824, 0.3325258 , 0.13162416, 0.64291933, 0.12136688,
0.25679033, 0.15446923, 0.09326367, 0.2111585 , 0.35136293,
0.17856622, 0.11747726, 0.14689857, 0.91334888, 0.33472757,
0.61877994, 0.16414568, 0.62126456, 0.16774844, 0.85266978,
0.89655915, 0.16545898, 0.24611915, 0.16006513, 0.70327315,
0.15827211, 0.85634043, 0.12054177, 0.1375202 , 0.56988285,
0.10391274, 0.87788259, 0.86962862, 0.1305029 , 0.92015643,
0.15582768, 0.1309077 , 0.53155835, 0.89622127, 0.17553661,
0.15585283, 0.90915277, 0.16579567, 0.13122589, 0.87592006,
0.90842303, 0.48844404, 0.17292445, 0.19909204, 0.13524414,
0.1375202 , 0.13987027, 0.53906011, 0.5941846 , 0.16075476,
0.83407033, 0.12398808, 0.11946654, 0.14628144, 0.18797311,
0.38986125, 0.87810898, 0.56353165, 0.12784258, 0.1030095 ,
0.91276766, 0.14226138, 0.88795149, 0.12588196, 0.12914704,
0.90748673, 0.12667147, 0.91041256, 0.36696583, 0.3078374 ,
0.19327157, 0.15249874, 0.26395095, 0.65418889, 0.64831127,
0.6541947 , 0.90753918, 0.56786185, 0.12962877, 0.85998813,
0.10048191, 0.12962877, 0.41631829])

In []: