Titanic Survival Predictions

December 12, 2020

Titanic Survival Predictions

Import pandas

[1]:
```
import pandas as pd
# optional
pd.set_option('display.max_columns',100)
pd.set_option('display.max_rows',100)
```

Load the clean titanic data

[2]:
```
train = pd.read_csv('clean_train.csv')
```

Inspect the data with `info()` and `head()`

[3]:
```
train.info()
train.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 24 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   891 non-null    int64
 1   Survived      891 non-null    int64
 2   Pclass        891 non-null    int64
 3   Name          891 non-null    object
 4   Sex           891 non-null    object
 5   Age           714 non-null    float64
 6   SibSp         891 non-null    int64
 7   Parch         891 non-null    int64
 8   Ticket        891 non-null    object
 9   Fare          891 non-null    float64
 10  Cabin         204 non-null    object
 11  Embarked      889 non-null    object
 12  Age_clean     891 non-null    float64
 13  Is_female     891 non-null    bool
 14  Emb_C         891 non-null    int64
 15  Emb_Q         891 non-null    int64
 16  Emb_S         891 non-null    int64
```

```
 17  Fare_usd     891 non-null    float64
 18  Family_size  891 non-null    int64
 19  Title        891 non-null    object
 20  Mr           891 non-null    int64
 21  Mrs          891 non-null    int64
 22  Miss         891 non-null    int64
 23  Master       891 non-null    int64
dtypes: bool(1), float64(4), int64(13), object(6)
memory usage: 161.1+ KB
```

[3]:

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Age_clean  Is_female  \
0      0         A/5 21171   7.2500   NaN        S       22.0      False
1      0          PC 17599  71.2833   C85        C       38.0       True
2      0  STON/O2. 3101282   7.9250   NaN        S       26.0       True
3      0            113803  53.1000  C123        S       35.0       True
4      0            373450   8.0500   NaN        S       35.0      False

   Emb_C  Emb_Q  Emb_S  Fare_usd  Family_size Title  Mr  Mrs  Miss  Master
0      0      0      1   9.42500            1    Mr   1    0     0       0
1      1      0      0  92.66829            1   Mrs   0    1     0       0
2      0      0      1  10.30250            0  Miss   0    0     1       0
3      0      0      1  69.03000            1   Mrs   0    1     0       0
4      0      0      1  10.46500            0    Mr   1    0     0       0
```

Create two variables for features (Age_clean, Is_female, Pclass, emb_C, emb_Q, emb_S, Fare) and target (Survived).

[4]:

```
features = train[['Age_clean', 'Is_female',␣
 ↪'Pclass','Emb_C','Emb_Q','Emb_S','Fare']]
target = train.Survived
```

[5]:

```
# Ensure data is prepped correctly
print(features.shape)
target.shape
```

```
(891, 7)
```

[5]:

```
(891,)
```

Import `train_test_split` from `sklearn.model_selection`

[6]:

```python
from sklearn.model_selection import train_test_split
```

Split the titanic data into training sets and test sets

[7]:

```python
features_train, features_test, target_train, target_test = train_test_split(features, target, test_size=.2, random_state=0)
```

[8]:

```python
print(features_train.shape, features_test.shape)
print(target_train.shape, target_test.shape)
```

```
(712, 7) (179, 7)
(712,) (179,)
```

[9]:

```python
features_train.head()
```

[9]:

	Age_clean	Is_female	Pclass	Emb_C	Emb_Q	Emb_S	Fare
140	29.699118	True	3	1	0	0	15.2458
439	31.000000	False	2	0	0	1	10.5000
817	31.000000	False	2	1	0	0	37.0042
378	20.000000	False	3	1	0	0	4.0125
491	21.000000	False	3	0	0	1	7.2500

[10]:

```python
target_train.head()
```

[10]:

```
140    0
439    0
817    0
378    0
491    0
Name: Survived, dtype: int64
```

Import the `KNeighborsClassifier` from `sklearn.neighbors`

[11]:

```python
from sklearn.neighbors import KNeighborsClassifier
```

Create and train a KNN model

[12]:

```python
knneighbors = KNeighborsClassifier(n_neighbors=5, weights='uniform')
knneighbors.fit(features_train, target_train)
```

[12]:

```
KNeighborsClassifier()
```

Make predictions with test data

[13]:
```python
print(knneighbors.predict(features_test))
print(target_test.values)
```

```
[1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0
 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0
 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0]
[0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0
 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 1 1 1 1 0 0
 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0
 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1
 1 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0]
```

Predict probabilities

[14]:
```python
knneighbors.predict_proba(features_test)
```

[14]:
```
array([[0.4, 0.6],
       [0.8, 0.2],
       [0.8, 0.2],
       [0.4, 0.6],
       [0.6, 0.4],
       [0.6, 0.4],
       [0.4, 0.6],
       [0.4, 0.6],
       [0. , 1. ],
       [0.6, 0.4],
       [1. , 0. ],
       [0.8, 0.2],
       [0.8, 0.2],
       [0.6, 0.4],
       [0.4, 0.6],
       [0.2, 0.8],
       [1. , 0. ],
       [0.6, 0.4],
       [1. , 0. ],
       [0.2, 0.8],
       [0.6, 0.4],
       [0.8, 0.2],
       [1. , 0. ],
       [0.6, 0.4],
       [0.8, 0.2],
       [0. , 1. ],
       [0.4, 0.6],
       [0.4, 0.6],
       [0.4, 0.6],
       [0.6, 0.4],
```

```
[1. , 0. ],
[0.8, 0.2],
[0.6, 0.4],
[0.6, 0.4],
[1. , 0. ],
[0.8, 0.2],
[0.8, 0.2],
[0.4, 0.6],
[0.8, 0.2],
[0.8, 0.2],
[0.2, 0.8],
[1. , 0. ],
[0.6, 0.4],
[0.6, 0.4],
[0.2, 0.8],
[1. , 0. ],
[1. , 0. ],
[0.4, 0.6],
[1. , 0. ],
[0.2, 0.8],
[0.8, 0.2],
[0.2, 0.8],
[0.4, 0.6],
[0.6, 0.4],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[1. , 0. ],
[0.2, 0.8],
[0.8, 0.2],
[0.2, 0.8],
[0.6, 0.4],
[0.2, 0.8],
[0.6, 0.4],
[0.4, 0.6],
[0.2, 0.8],
[0.4, 0.6],
[0.4, 0.6],
[0.2, 0.8],
[1. , 0. ],
[1. , 0. ],
[0.8, 0.2],
[0.4, 0.6],
```

```
[0.8, 0.2],
[1. , 0. ],
[0.4, 0.6],
[0.6, 0.4],
[0.4, 0.6],
[0.2, 0.8],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[0.4, 0.6],
[1. , 0. ],
[1. , 0. ],
[0. , 1. ],
[1. , 0. ],
[0.6, 0.4],
[0.4, 0.6],
[0.2, 0.8],
[0.8, 0.2],
[0.8, 0.2],
[0.6, 0.4],
[0.6, 0.4],
[0.8, 0.2],
[0.2, 0.8],
[0.8, 0.2],
[0.4, 0.6],
[0.8, 0.2],
[0.8, 0.2],
[0.6, 0.4],
[0.6, 0.4],
[0.6, 0.4],
[0.6, 0.4],
[0.4, 0.6],
[0.6, 0.4],
[1. , 0. ],
[0.2, 0.8],
[0.2, 0.8],
[1. , 0. ],
[0. , 1. ],
[0.6, 0.4],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[0.8, 0.2],
[0.4, 0.6],
[0.8, 0.2],
[0.8, 0.2],
[1. , 0. ],
```

```
[1. , 0. ],
[0.8, 0.2],
[1. , 0. ],
[1. , 0. ],
[1. , 0. ],
[0.8, 0.2],
[0.4, 0.6],
[0.8, 0.2],
[1. , 0. ],
[1. , 0. ],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
[0.6, 0.4],
[0. , 1. ],
[1. , 0. ],
[0.6, 0.4],
[0.6, 0.4],
[1. , 0. ],
[0.4, 0.6],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[0.4, 0.6],
[0.4, 0.6],
[1. , 0. ],
[0.8, 0.2],
[0.6, 0.4],
[0.6, 0.4],
[0.6, 0.4],
[1. , 0. ],
[0.2, 0.8],
[0.2, 0.8],
[1. , 0. ],
[0.8, 0.2],
[0.6, 0.4],
[0.8, 0.2],
[0.4, 0.6],
[1. , 0. ],
[0.8, 0.2],
[1. , 0. ],
[0.8, 0.2],
[0.8, 0.2],
[0.8, 0.2],
```

```
        [0.4, 0.6],
        [1. , 0. ],
        [0.6, 0.4],
        [0. , 1. ],
        [1. , 0. ],
        [0. , 1. ],
        [1. , 0. ],
        [0.6, 0.4]])
```

Check the classes of the model

[15]:
```
knneighbors.classes_
```

[15]:
```
array([0, 1])
```

Score the model

[16]:
```
knneighbors.score(features_test, target_test)
```

[16]:
```
0.7206703910614525
```

0.0.1 Stratified KFolds Crossvalidation

Import `cross_val_score,StratifiedKFold,RepeatedStratifiedKFold` from `sklearn.model_selection`

[17]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold,␣
↪RepeatedStratifiedKFold
```

Use `cross_val_score` to evaluate model performance

[18]:
```
results = cross_val_score(KNeighborsClassifier(n_neighbors=5,␣
↪weights='uniform'),
                          features,
                          target)
results.mean()
```

[18]:
```
0.6858200991777037
```

Use `StratifiedKFold` to shuffle the data

[19]:
```
results1 = cross_val_score(KNeighborsClassifier(n_neighbors=5,␣
↪weights='uniform'),
                          features,
                          target,
                          cv=StratifiedKFold(shuffle=True, random_state=0))
results1.mean()
```

[19]:
```
0.7037285794990897
```

Use `RepeatedStratifiedKFold` to calculate multiple iterations of crossvalidation

[20]:
```
results2 = cross_val_score(KNeighborsClassifier(n_neighbors=5,␣
↪weights='uniform'),
                          features,
                          target,
                          cv=RepeatedStratifiedKFold(n_splits=5, n_repeats=10,␣
↪random_state=0))
results2.mean()
results2
```

[20]:
```
array([0.68156425, 0.65168539, 0.70786517, 0.75280899, 0.7247191 ,
       0.67039106, 0.65730337, 0.62921348, 0.67977528, 0.75842697,
       0.75977654, 0.65730337, 0.66292135, 0.71910112, 0.71348315,
       0.67039106, 0.69101124, 0.69101124, 0.69101124, 0.69101124,
       0.67597765, 0.6741573 , 0.70786517, 0.6741573 , 0.69101124,
       0.67039106, 0.75280899, 0.66853933, 0.71348315, 0.71910112,
       0.69832402, 0.61797753, 0.75280899, 0.67977528, 0.6741573 ,
       0.68715084, 0.68539326, 0.67977528, 0.69662921, 0.69662921,
       0.68156425, 0.69662921, 0.74719101, 0.71348315, 0.68539326,
       0.68156425, 0.69101124, 0.71910112, 0.70786517, 0.63483146])
```

Create a for loop to score both uniform and distance weighted models for different values of k to determine optimal model parameters.

[21]:
```
# Create lists to store data for data frame
neighbors, scores_distance, scores_uniform = [],[],[]

for k in range (1, 101):
    score_u = cross_val_score(KNeighborsClassifier(n_neighbors=k,␣
↪weights='uniform'),
                          features,
                          target,
                          cv=StratifiedKFold(shuffle=True, random_state=0))
    scores_uniform.append(score_u.mean())
    score_d = cross_val_score(KNeighborsClassifier(n_neighbors=k,␣
↪weights='distance'),
                          features,
                          target,
                          cv=StratifiedKFold(shuffle=True, random_state=0))
    scores_distance.append(score_d.mean())
    neighbors.append(k)
print(scores_uniform)
print(scores_distance)
print(neighbors)
```

```
[0.6913439206578369, 0.694739815454146, 0.7014876655577177, 0.694739815454146,
0.7037285794990897, 0.6812692235264579, 0.6992341974766179, 0.6969744523256542,
```

```
0.7003201305630531, 0.7081852991023789, 0.7070554265268972, 0.703690917079907,
0.7115247002699141, 0.7059255539514154, 0.7036658088004519, 0.7025610445044252,
0.7047956813759337, 0.7025735986441528, 0.7093026175381332, 0.7014500031385349,
0.7003138534931894, 0.7059255539514154, 0.7048019584457975, 0.7070428723871697,
0.6991839809177077, 0.6991777038478437, 0.6980290000627707, 0.6946644906157806,
0.6991588726382525, 0.69580691733099, 0.6946770447555082, 0.696930512836608,
0.6868181532860461, 0.6868244303559099, 0.6868181532860461, 0.6868307074257736,
0.6901952168727638, 0.6879543029313917, 0.687941748791664, 0.684577239344674,
0.6868369844956375, 0.6812127298976838, 0.6812127298976838, 0.6812127298976838,
0.6778419433808298, 0.6868118762161823, 0.6811938986880924, 0.6755884752997302,
0.6800828573222021, 0.684577239344674, 0.683453643839056, 0.682330048333438,
0.6789592618165841, 0.6778419433808298, 0.6834599209089198, 0.6767183478752118,
0.6767120708053482, 0.6789592618165841, 0.6789592618165841, 0.6823363254033017,
0.6800891343920658, 0.6789655388864478, 0.6778419433808298, 0.6711129244868494,
0.6722302429226037, 0.6722365199924676, 0.6755947523695939, 0.6733475613583579,
0.6789655388864478, 0.6744774339338397, 0.6767246249450756, 0.6767309020149395,
0.6778544975205574, 0.6789843700960392, 0.676743456154667, 0.6801079656016571,
0.6733601154980855, 0.6744837110037034, 0.6733538384282218, 0.6722302429226037,
0.6755947523695939, 0.6621367145816334, 0.6643713514531416, 0.6677421379699957,
0.6756073065093214, 0.6688720105454773, 0.6733601154980855, 0.6688720105454774,
0.6756073065093215, 0.6733663925679493, 0.6711192015567133, 0.6711192015567133,
0.6722365199924676, 0.6711129244868494, 0.6722365199924676, 0.6733601154980855,
0.6699893289812315, 0.6677609691795869, 0.6677672462494507, 0.668884564685205]
[0.6913439206578369, 0.6980603854120897, 0.7160316364321135, 0.7149080409264954,
0.7250078463373296, 0.7250141234071935, 0.7216370598204758, 0.7373485656895361,
0.7249952921976022, 0.7306007155859644, 0.7272487602787019, 0.7261188877032201,
0.7306007155859644, 0.7317243110915824, 0.7294645659406189, 0.7272362061389744,
0.7272299290691105, 0.7238654196221204, 0.7283535245747286, 0.7249890151277384,
0.7283535245747286, 0.7317243110915823, 0.7306069926558282, 0.7272487602787019,
0.7305944385161006, 0.7294708430104827, 0.7294645659406189, 0.729458288870755,
0.733952670893227, 0.7339463938233632, 0.7328290753876091, 0.7317054798819912,
0.7328227983177453, 0.7294520118008914, 0.7350637122591175, 0.7316992028121274,
0.7361873077647354, 0.7316992028121272, 0.7373109032703533, 0.7350637122591175,
0.735076266398845, 0.7339589479630908, 0.733952670893227, 0.7317054798819911,
0.7317054798819911, 0.7339463938233632, 0.7339463938233632, 0.7328227983177452,
0.7350825434687087, 0.732829075387609, 0.733952670893227, 0.7361998619044631,
0.7361998619044631, 0.7339589479630908, 0.731711756951855, 0.730588161446237,
0.7305818843763732, 0.7328227983177452, 0.7305881614462368, 0.7305881614462368,
0.731711756951855, 0.733952670893227, 0.728340970435001, 0.7283409704350008,
0.7317180340217186, 0.7294645659406189, 0.7317117569518549, 0.730588161446237,
0.7328227983177453, 0.7316992028121273, 0.7316992028121274, 0.732829075387609,
0.7317054798819911, 0.7305881614462368, 0.7283472475048647, 0.7261000564936289,
0.7249764609880108, 0.7249764609880108, 0.7272236519992468, 0.7261000564936289,
0.7261000564936287, 0.7261000564936289, 0.7294582888707553, 0.7294645659406189,
0.7294582888707553, 0.7283346933651373, 0.7272110978595192, 0.7272173749293829,
0.7283346933651372, 0.726093779423765, 0.7283346933651372, 0.727217374929383,
0.7272110978595192, 0.7283409704350009, 0.723846588412529, 0.724970183918147,
0.723840311342665 2, 0.7249639068482832, 0.7215931203314293, 0.722722992906911]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

Store the model scores in a DataFrame

[22]:
```
knneighbors_scores_dict = {
    'distance_weighted':scores_distance,
    'uniform_weighted':scores_uniform
}

knneighbors_scores = pd.DataFrame(knneighbors_scores_dict, index =neighbors)
knneighbors_scores
```

[22]:

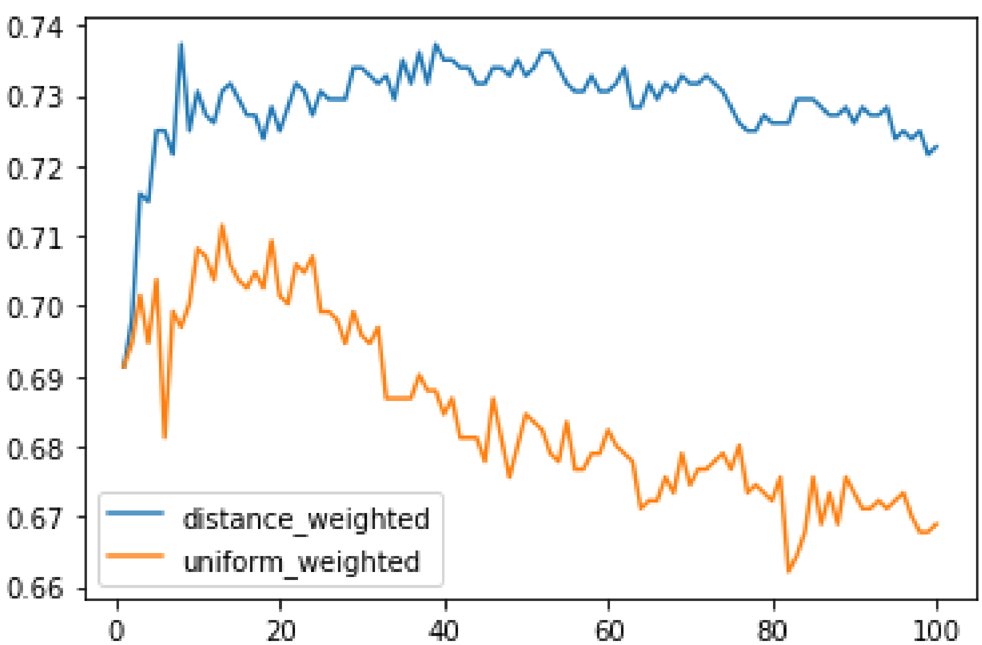
	distance_weighted	uniform_weighted
1	0.691344	0.691344
2	0.698060	0.694740
3	0.716032	0.701488
4	0.714908	0.694740
5	0.725008	0.703729
6	0.725014	0.681269
7	0.721637	0.699234
8	0.737349	0.696974
9	0.724995	0.700320
10	0.730601	0.708185
11	0.727249	0.707055
12	0.726119	0.703691
13	0.730601	0.711525
14	0.731724	0.705926
15	0.729465	0.703666
16	0.727236	0.702561
17	0.727230	0.704796
18	0.723865	0.702574
19	0.728354	0.709303
20	0.724989	0.701450
21	0.728354	0.700314
22	0.731724	0.705926
23	0.730607	0.704802
24	0.727249	0.707043
25	0.730594	0.699184
26	0.729471	0.699178
27	0.729465	0.698029
28	0.729458	0.694664
29	0.733953	0.699159
30	0.733946	0.695807
31	0.732829	0.694677

32	0.731705	0.696931
33	0.732823	0.686818
34	0.729452	0.686824
35	0.735064	0.686818
36	0.731699	0.686831
37	0.736187	0.690195
38	0.731699	0.687954
39	0.737311	0.687942
40	0.735064	0.684577
41	0.735076	0.686837
42	0.733959	0.681213
43	0.733953	0.681213
44	0.731705	0.681213
45	0.731705	0.677842
46	0.733946	0.686812
47	0.733946	0.681194
48	0.732823	0.675588
49	0.735083	0.680083
50	0.732829	0.684577
51	0.733953	0.683454
52	0.736200	0.682330
53	0.736200	0.678959
54	0.733959	0.677842
55	0.731712	0.683460
56	0.730588	0.676718
57	0.730582	0.676712
58	0.732823	0.678959
59	0.730588	0.678959
60	0.730588	0.682336
61	0.731712	0.680089
62	0.733953	0.678966
63	0.728341	0.677842
64	0.728341	0.671113
65	0.731718	0.672230
66	0.729465	0.672237
67	0.731712	0.675595
68	0.730588	0.673348
69	0.732823	0.678966
70	0.731699	0.674477
71	0.731699	0.676725
72	0.732829	0.676731
73	0.731705	0.677854
74	0.730588	0.678984
75	0.728347	0.676743
76	0.726100	0.680108
77	0.724976	0.673360
78	0.724976	0.674484

79	0.727224	0.673354
80	0.726100	0.672230
81	0.726100	0.675595
82	0.726100	0.662137
83	0.729458	0.664371
84	0.729465	0.667742
85	0.729458	0.675607
86	0.728335	0.668872
87	0.727211	0.673360
88	0.727217	0.668872
89	0.728335	0.675607
90	0.726094	0.673366
91	0.728335	0.671119
92	0.727217	0.671119
93	0.727211	0.672237
94	0.728341	0.671113
95	0.723847	0.672237
96	0.724970	0.673360
97	0.723840	0.669989
98	0.724964	0.667761
99	0.721593	0.667767
100	0.722723	0.668885

[23]:
```
knneighbors_scores.plot.line()
```

[23]: <matplotlib.axes._subplots.AxesSubplot at 0x119cac7c0>



0.1 Data Normalization

Calculate the mean and standard deviation of Age_clean and Fare

[24]:
```
avg_age = train.Age_clean.mean()
std_age = train.Age_clean.std()
```

[25]:
```
avg_fare = train.Fare.mean()
std_fare = train.Fare.std()
```

Subtract the mean and divide by the standard deviation

[26]:
```
train['Age_normalized'] = (train.Age_clean - avg_age) / std_age
train['Fare_normalized'] = (train.Fare - avg_fare) / std_fare
```

Create new features using normalized data (Age_norm, Is_female, Pclass, emb_C, emb_Q, emb_S, Fare_norm)

[27]:
```
features_normalized = train[['Age_normalized', 'Is_female',␣
↪'Pclass','Emb_C','Emb_Q','Emb_S','Fare_normalized']]
```

Score the KNN on normalized data with a for loop comparing different numbers of neighbors

[28]:
```
for k in range (25,51):
    print(k, '-', cross_val_score(KNeighborsClassifier(n_neighbors=k,␣
↪weights='distance'),
                             features_normalized,
                             target,
                             cv=StratifiedKFold(shuffle=True)).mean())
```

```
25 - 0.8058062896240035
26 - 0.8113866047329106
27 - 0.797991337643588
28 - 0.7957315924926245
29 - 0.804670139978658
30 - 0.8002448057246877
31 - 0.8013495700207143
32 - 0.7946017199171427
33 - 0.8036030381018142
34 - 0.7901136149645346
35 - 0.7856506182913815
36 - 0.7946017199171427
37 - 0.8002134203753688
38 - 0.8035967610319503
39 - 0.7991086560793421
40 - 0.8002510827945514
41 - 0.7957441466323522
```

42 - 0.7900885066850794
43 - 0.7822735547046639
44 - 0.7946142740568704
45 - 0.8036093151716781
46 - 0.8013935095097608
47 - 0.8047266336074321
48 - 0.7957315924926245
49 - 0.8058125666938671
50 - 0.8002259745150964

0.2 Random Forest

Import the `RandomForestClassifier` from `sklearn.ensemble`

[29]:
```
from sklearn.ensemble import RandomForestClassifier
```

Create a model and train it on the training sets

[30]:
```
randomforest = RandomForestClassifier()
randomforest.fit(features, target)
```

[30]:
```
RandomForestClassifier()
```

Calculate the score for the Random Forest model using the test sets

[31]:
```
randomforest.feature_importances_
```

[31]:
```
array([0.29588566, 0.26139002, 0.09298923, 0.01347759, 0.00726209,
       0.01345015, 0.31554527])
```

[32]:
```
features.columns
```

[32]:
```
Index(['Age_clean', 'Is_female', 'Pclass', 'Emb_C', 'Emb_Q', 'Emb_S', 'Fare'],
dtype='object')
```

[33]:
```
randomforest_results = cross_val_score(RandomForestClassifier(),
                                       features,
                                       target,
                                       cv=RepeatedStratifiedKFold(n_splits=5,␣
 ↪n_repeats=10, random_state=0))
randomforest_results
```

[33]:
```
array([0.81564246, 0.8258427 , 0.83146067, 0.80898876, 0.79775281,
       0.82122905, 0.83146067, 0.8258427 , 0.79213483, 0.8258427 ,
       0.80446927, 0.83707865, 0.78089888, 0.83146067, 0.83146067,
       0.77653631, 0.80337079, 0.83707865, 0.84269663, 0.82022472,
       0.82681564, 0.80337079, 0.82022472, 0.78651685, 0.8258427 ,
       0.83240223, 0.85955056, 0.78651685, 0.78089888, 0.8258427 ,
       0.79329609, 0.79775281, 0.80337079, 0.81460674, 0.81460674,
```

```
         0.78212291, 0.78089888, 0.84269663, 0.84269663, 0.84269663,
         0.80446927, 0.80337079, 0.80898876, 0.80898876, 0.83146067,
         0.82681564, 0.84831461, 0.83146067, 0.8258427 , 0.78089888])
```

[34]:
```
randomforest_results.mean()
```

[34]:
```
0.8154962023727325
```

Add more features to the model to try and improve the performance of your model___

[35]:
```
features2 = train[['Age_clean', 'Is_female', 'Pclass',␣
 ↪'Fare','Family_size','Mr','Mrs','Miss','Master']]
```

[36]:
```
randomforest.fit(features2, target)
randomforest.feature_importances_
```

[36]:
```
array([0.23771512, 0.12989556, 0.07947193, 0.26233483, 0.0851964 ,
       0.14678425, 0.02710502, 0.02095985, 0.01053705])
```

[37]:
```
features2.columns
```

[37]:
```
Index(['Age_clean', 'Is_female', 'Pclass', 'Fare', 'Family_size', 'Mr', 'Mrs',
       'Miss', 'Master'],
      dtype='object')
```

[38]:
```
features3 = train[['Age_clean', 'Is_female', 'Pclass',␣
 ↪'Fare','Family_size','Mr']]
results3 = cross_val_score(RandomForestClassifier(),
                                      features3,
                                      target,
                                      cv=RepeatedStratifiedKFold(n_splits=5,␣
 ↪n_repeats=10))
results3
```

[38]:
```
array([0.79888268, 0.78651685, 0.81460674, 0.8258427 , 0.83146067,
       0.7877095 , 0.82022472, 0.82022472, 0.80337079, 0.80898876,
       0.83798883, 0.84831461, 0.79213483, 0.7752809 , 0.85955056,
       0.80446927, 0.84831461, 0.80337079, 0.7752809 , 0.86516854,
       0.80446927, 0.83707865, 0.79775281, 0.83707865, 0.85955056,
       0.83240223, 0.8258427 , 0.83707865, 0.83146067, 0.79775281,
       0.88826816, 0.83146067, 0.80898876, 0.84269663, 0.80898876,
       0.77094972, 0.80898876, 0.78651685, 0.83707865, 0.79213483,
       0.83240223, 0.82022472, 0.82022472, 0.85393258, 0.81460674,
       0.79888268, 0.81460674, 0.8258427 , 0.80337079, 0.84831461])
```

[39]:
```
results3.mean()
```

[39]: 0.8195329860021342

[40]:
```
features4 = train[['Age_clean', 'Is_female', 'Fare','Family_size']]
randomforest.fit(features4, target)
randomforest.feature_importances_
```

[40]: array([0.26991764, 0.26935093, 0.35936   , 0.10137143])

[41]:
```
results4 = cross_val_score(RandomForestClassifier(),
                                       features4,
                                       target,
                                       cv=RepeatedStratifiedKFold(n_splits=5,␣
 ↪n_repeats=10))
results4
```

[41]:
```
array([0.84916201, 0.80337079, 0.8258427 , 0.84831461, 0.75842697,
       0.82122905, 0.80898876, 0.84269663, 0.85393258, 0.78651685,
       0.81564246, 0.83707865, 0.78089888, 0.78089888, 0.86516854,
       0.79888268, 0.80898876, 0.80898876, 0.80337079, 0.81460674,
       0.82681564, 0.78089888, 0.83146067, 0.82022472, 0.82022472,
       0.74860335, 0.86516854, 0.80898876, 0.84269663, 0.84831461,
       0.79329609, 0.8258427 , 0.78651685, 0.84269663, 0.85955056,
       0.82681564, 0.82022472, 0.84269663, 0.76404494, 0.8258427 ,
       0.81564246, 0.78651685, 0.85955056, 0.80898876, 0.8258427 ,
       0.82681564, 0.83146067, 0.79775281, 0.78089888, 0.80898876])
```

[42]:
```
results4.mean()
```

[42]: 0.8167277634800075

0.2.1 Make a Prediction for Kaggle

Train the model using the best data available

[43]:
```
train_features_final = train[['Age_clean', 'Is_female', 'Pclass',␣
 ↪'Fare','Family_size','Mr']]
```

[45]:
```
randomforest_final = RandomForestClassifier().fit(train_features_final, target)
```

Create a DataFrame with the clean_test.csv data

[46]:
```
test = pd.read_csv('clean_test.csv')
```

Investigate the test data

[47]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 22 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         418 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
 11  Age_clean    418 non-null    float64
 12  Is_female    418 non-null    bool
 13  Emb_C        418 non-null    int64
 14  Emb_Q        418 non-null    int64
 15  Emb_S        418 non-null    int64
 16  Family_size  418 non-null    int64
 17  Title        418 non-null    object
 18  Mr           418 non-null    int64
 19  Mrs          418 non-null    int64
 20  Miss         418 non-null    int64
 21  Master       418 non-null    int64
dtypes: bool(1), float64(3), int64(12), object(6)
memory usage: 69.1+ KB
```

[48]:
```
test.head()
```

[48]:
```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked  Age_clean  Is_female  \
0  34.5      0      0   330911   7.8292   NaN        Q       34.5      False
1  47.0      1      0   363272   7.0000   NaN        S       47.0       True
2  62.0      0      0   240276   9.6875   NaN        Q       62.0      False
3  27.0      0      0   315154   8.6625   NaN        S       27.0      False
4  22.0      1      1  3101298  12.2875   NaN        S       22.0       True

   Emb_C  Emb_Q  Emb_S  Family_size Title  Mr  Mrs  Miss  Master
```

```
0        0        1        0                0     Mr    1      0      0         0
1        0        0        1                1    Mrs    0      1      0         0
2        0        1        0                0     Mr    1      0      0         0
3        0        0        1                0     Mr    1      0      0         0
4        0        0        1                2    Mrs    0      1      0         0
```

Select the same features from test data (Age_clean, Is_female, Pclass, emb_C, emb_Q, emb_S, Fare)

[49]:
```python
test_features_final = test[['Age_clean', 'Is_female', 'Pclass',␣
↪'Fare','Family_size','Mr']]
```

Create predictions

[50]:
```python
predictions_final = randomforest_final.predict(test_features_final)
predictions_final
```

[50]:
```
array([0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1])
```

Add Predictions as a new column (Survived) in the DataFrame with the test data

[51]:
```python
test['Survived'] = predictions_final
test.head()
```

[51]:
```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female
```

```
    Age  SibSp  Parch   Ticket     Fare Cabin Embarked  Age_clean  Is_female  \
0  34.5      0      0   330911   7.8292   NaN        Q       34.5      False
1  47.0      1      0   363272   7.0000   NaN        S       47.0       True
2  62.0      0      0   240276   9.6875   NaN        Q       62.0      False
3  27.0      0      0   315154   8.6625   NaN        S       27.0      False
4  22.0      1      1  3101298  12.2875   NaN        S       22.0       True

   Emb_C  Emb_Q  Emb_S  Family_size Title  Mr  Mrs  Miss  Master  Survived
0      0      1      0            0    Mr   1    0     0       0         0
1      0      0      1            1   Mrs   0    1     0       0         0
2      0      1      0            0    Mr   1    0     0       0         1
3      0      0      1            0    Mr   1    0     0       0         1
4      0      0      1            2   Mrs   0    1     0       0         1
```

Save as CSV (make sure you set index=False)

[52]:
```python
kaggle = test[['PassengerId','Survived']]
kaggle.head()
```

[52]:
```
   PassengerId  Survived
0          892         0
1          893         0
2          894         1
3          895         1
4          896         1
```

[53]:
```python
kaggle.to_csv('titanic_survival_predictions.csv', index=False)
```