# Untitled

February 28, 2025

[362]:
```
import os
os.environ["OMP_NUM_THREADS"] = "3"
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from scipy.stats import fisher_exact
from scipy.stats import MonteCarloMethod
import statsmodels.api as sm
```

[363]:
```
os.chdir('C://Users/ordav/Desktop/ML_Projects/titanic')
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
testOrg = pd.read_csv('test.csv')
```

[364]:
```
train.head()
```

[364]:

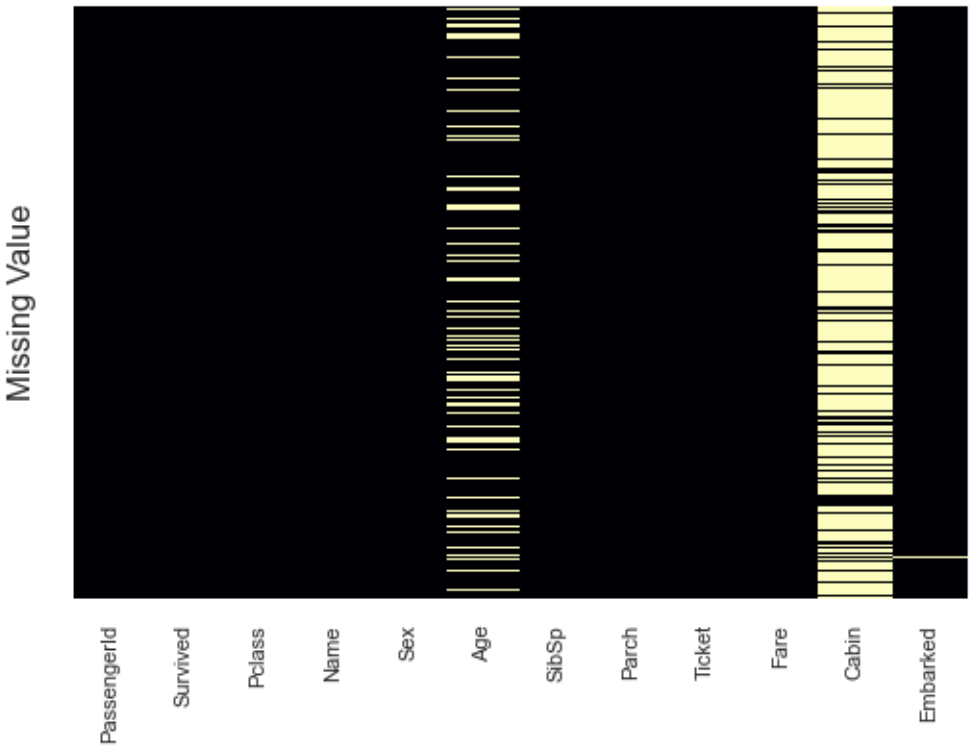
| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th… | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

[365]:
```
train.describe()
```

[365]:

| | PassengerId | Survived | Pclass | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| count | 891.000000 | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| mean | 446.000000 | 0.383838 | 2.308642 | 29.699118 | 0.523008 | 0.381594 | 32.204208 |
| std | 257.353842 | 0.486592 | 0.836071 | 14.526497 | 1.102743 | 0.806057 | 49.693429 |
| min | 1.000000 | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 223.500000 | 0.000000 | 2.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 |
| 50% | 446.000000 | 0.000000 | 3.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 |
| 75% | 668.500000 | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| max | 891.000000 | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

[366]:
```
sns.set(rc={'figure.figsize':(6, 4)})
NAsHeatmap = sns.heatmap(train.isnull(), cmap = 'magma', cbar=False)
plt.xticks(rotation=90, fontsize=8)
NAsHeatmap.set(yticklabels = [])
NAsHeatmap.set_ylabel('Missing Value')
NAsHeatmap.tick_params(left=False)
```




[367]:
```
train = train.drop(['Ticket'], axis = 1)
test_id = test['PassengerId']
test = test.drop(['Ticket'], axis = 1)
```

[368]:
```
train.isna().sum()
```

[368]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

[369]:
```
test.isna().sum()
```

[369]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

[370]:
```
train = train.drop('Cabin', axis = 1)
test = test.drop('Cabin', axis = 1)
```

[371]:
```
train.loc[train.Embarked.isna(), :]
```

[371]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Fare | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|
| 61 | 62 | 1 | 1 | Icard, Miss. Amelie | female | 38.0 | 0 | 0 | 80.0 | NaN |
| 829 | 830 | 1 | 1 | Stone, Mrs. George Nelson (Martha Evelyn) | female | 62.0 | 0 | 0 | 80.0 | NaN |

[372]:
```
np.sum(train['Fare'] == 0)
```

[372]:
```
15
```

[373]:
```
train['Fare'] = train['Fare'].replace(0, value = np.nan)
test['Fare'] = test['Fare'].replace(0, value = np.nan)
```

[374]:
```
train.loc[:, 'Embarked'] = train.loc[:, 'Embarked'].fillna(train.loc[train.
 ↪Pclass == 1, 'Embarked'].mode()[0])
```

[375]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
encoded_data = encoder.fit_transform(train[['Embarked']])
train = train.join(pd.DataFrame(encoded_data.toarray(), columns = encoder.
 ↪get_feature_names_out()).iloc[:, [1, 2]])

encoder = OneHotEncoder()
encoded_data = encoder.fit_transform(test[['Embarked']])
test = test.join(pd.DataFrame(encoded_data.toarray(), columns = encoder.
 ↪get_feature_names_out()).iloc[:, [1, 2]])
```

[376]:
```
encoder = OneHotEncoder()
encoded_data = encoder.fit_transform(train[['Sex']])
train = train.join(pd.DataFrame(encoded_data.toarray(), columns = encoder.
 ↪get_feature_names_out()).iloc[:, 1])

encoder = OneHotEncoder()
encoded_data = encoder.fit_transform(test[['Sex']])
test = test.join(pd.DataFrame(encoded_data.toarray(), columns = encoder.
 ↪get_feature_names_out()).iloc[:, 1])
```

[377]:
```
EmbarkedColTrain = train.Embarked
EmbarkedColTest = test.Embarked

SexColTrain = train.Sex
SexColTest = test.Sex

NameColTrain = train.Name
NameColTest = test.Name

IdColTrain = train.PassengerId
IdColTest = test.PassengerId

train.drop(['Embarked', 'Sex', 'Name', 'PassengerId'], inplace = True, axis = 1)
test.drop(['Embarked','Sex', 'Name', 'PassengerId'], inplace = True, axis = 1)
```

[378]:
```
y_train = train.Survived
train.drop('Survived', inplace = True, axis = 1)
```

[379]:
```
train
```

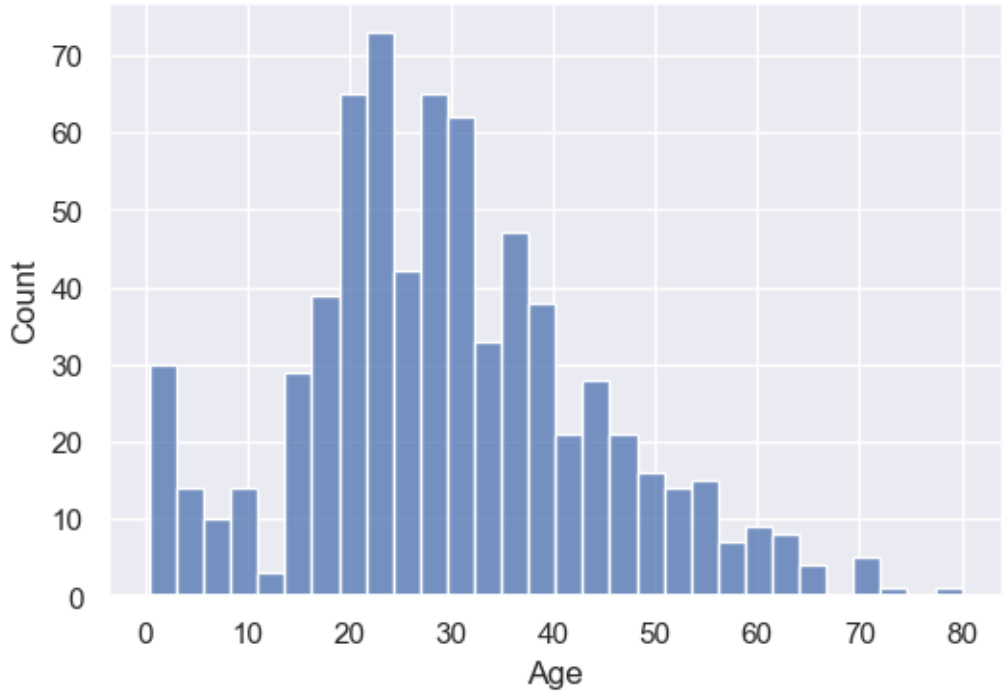
[379]:

| | Pclass | Age | SibSp | Parch | Fare | Embarked_Q | Embarked_S | Sex_male |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 22.0 | 1 | 0 | 7.2500 | 0.0 | 1.0 | 1.0 |
| 1 | 1 | 38.0 | 1 | 0 | 71.2833 | 0.0 | 0.0 | 0.0 |
| 2 | 3 | 26.0 | 0 | 0 | 7.9250 | 0.0 | 1.0 | 0.0 |
| 3 | 1 | 35.0 | 1 | 0 | 53.1000 | 0.0 | 1.0 | 0.0 |
| 4 | 3 | 35.0 | 0 | 0 | 8.0500 | 0.0 | 1.0 | 1.0 |
| .. | … | … | … | … | … | … | … | … |
| 886 | 2 | 27.0 | 0 | 0 | 13.0000 | 0.0 | 1.0 | 1.0 |
| 887 | 1 | 19.0 | 0 | 0 | 30.0000 | 0.0 | 1.0 | 0.0 |
| 888 | 3 | NaN | 1 | 2 | 23.4500 | 0.0 | 1.0 | 0.0 |
| 889 | 1 | 26.0 | 0 | 0 | 30.0000 | 0.0 | 0.0 | 1.0 |
| 890 | 3 | 32.0 | 0 | 0 | 7.7500 | 1.0 | 0.0 | 1.0 |

[891 rows x 8 columns]

[380]:
```
sns.histplot(x = 'Age', data = train, bins = 30)
```

[380]: <Axes: xlabel='Age', ylabel='Count'>



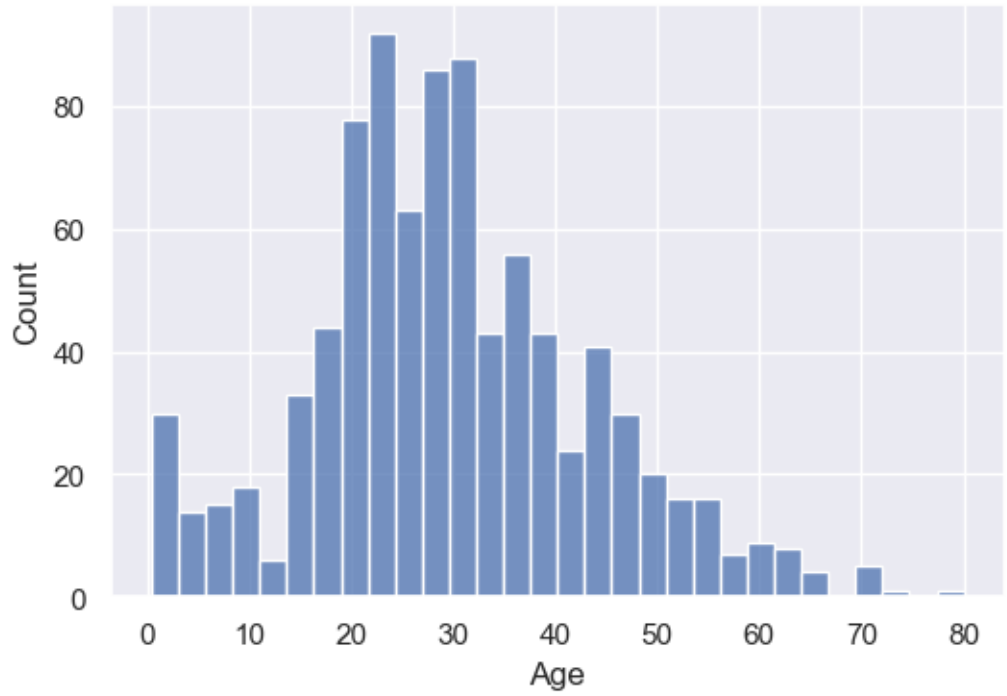

[381]:
```
X_train = train.copy()
X_test = test.copy()
```

[382]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor

### Using the imputer
rFor = RandomForestRegressor(n_estimators=300)
imputer = IterativeImputer(estimator=rFor, tol = 1)
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns = X_train.
 ↪columns)
X_test = pd.DataFrame(imputer.transform(X_test), columns = test.columns)
```

[383]:
```
train = X_train.copy()
test = X_test.copy()
```

[384]:
```
sns.histplot(x = 'Age', data = train, bins = 30)
```

[384]: <Axes: xlabel='Age', ylabel='Count'>




[385]:
```
train['Survived'] = y_train
train['Embarked'] = EmbarkedColTrain
test['Embarked'] = EmbarkedColTest
train['Name'] = NameColTrain
test['Name'] = NameColTest
train['PassengerId'] = IdColTrain
test['PassengerId'] = IdColTest

d = {0:'Female', 1:'Male'}

train['Sex'] = train.Sex_male.map(d)
test['Sex'] = test.Sex_male.map(d)

train.drop(['Embarked_Q', 'Embarked_S', 'Sex_male'], inplace = True, axis = 1)
test.drop(['Embarked_Q', 'Embarked_S', 'Sex_male'], inplace = True, axis = 1)
```

[386]:
```
def title(x):
    l = str.split(x)
    for j in range(len(l)):
        if l[j][-1] == ',':
            return l[j+1]

train['title'] = train.apply(lambda x: title(x['Name']), axis = 1)
test['title'] = test.apply(lambda x: title(x['Name']), axis = 1)
```

```
train.loc[(train.title != 'Mr.') & (train.title != 'Mrs.') & (train.title !=␣
 ↪'Miss.') & (train.title != 'Master.'), ['Name', 'title', 'PassengerId']]
```

[386]:

| | Name | title | PassengerId |
|---|---|---|---|
| 30 | Uruchurtu, Don. Manuel E | Don. | 31 |
| 149 | Byles, Rev. Thomas Roussel Davids | Rev. | 150 |
| 150 | Bateman, Rev. Robert James | Rev. | 151 |
| 245 | Minahan, Dr. William Edward | Dr. | 246 |
| 249 | Carter, Rev. Ernest Courtenay | Rev. | 250 |
| 317 | Moraweck, Dr. Ernest | Dr. | 318 |
| 369 | Aubart, Mme. Leontine Pauline | Mme. | 370 |
| 398 | Pain, Dr. Alfred | Dr. | 399 |
| 443 | Reynaldo, Ms. Encarnacion | Ms. | 444 |
| 449 | Peuchen, Major. Arthur Godfrey | Major. | 450 |
| 536 | Butt, Major. Archibald Willingham | Major. | 537 |
| 556 | Duff Gordon, Lady. (Lucille Christiana Sutherl… | Lady. | 557 |
| 599 | Duff Gordon, Sir. Cosmo Edmund ("Mr Morgan") | Sir. | 600 |
| 626 | Kirkland, Rev. Charles Leonard | Rev. | 627 |
| 632 | Stahelin-Maeglin, Dr. Max | Dr. | 633 |
| 641 | Sagesser, Mlle. Emma | Mlle. | 642 |
| 647 | Simonius-Blumer, Col. Oberst Alfons | Col. | 648 |
| 660 | Frauenthal, Dr. Henry William | Dr. | 661 |
| 694 | Weir, Col. John | Col. | 695 |
| 710 | Mayne, Mlle. Berthe Antonine ("Mrs de Villiers") | Mlle. | 711 |
| 745 | Crosby, Capt. Edward Gifford | Capt. | 746 |
| 759 | Rothes, the Countess. of (Lucy Noel Martha Dye… | the | 760 |
| 766 | Brewe, Dr. Arthur Jackson | Dr. | 767 |
| 796 | Leader, Dr. Alice (Farnham) | Dr. | 797 |
| 822 | Reuchlin, Jonkheer. John George | Jonkheer. | 823 |
| 848 | Harper, Rev. John | Rev. | 849 |
| 886 | Montvila, Rev. Juozas | Rev. | 887 |

[387]:
```
def TitleTrain(x,y):
    if x in ['Mr.', 'Mrs.', 'Miss.', 'Master.']:
        return x
    elif y  in [370 , 444, 557, 760]:
        return 'Mrs.'
    elif y in [642, 711]:
        return 'Miss.'
    else:
        return 'Mr.'

train['Title'] = train.apply(lambda x: TitleTrain(x['title'],␣
 ↪x['PassengerId']), axis = 1)
```

[388]:
```
test.loc[(test.title != 'Mr.') & (test.title != 'Mrs.') & (test.title != 'Miss.
↪') & (test.title != 'Master.'), ['Name', 'title', 'PassengerId']]
```

[388]:

| | Name | title | PassengerId |
|---|---|---|---|
| 88 | O'Donoghue, Ms. Bridget | Ms. | 980 |
| 131 | Gracie, Col. Archibald IV | Col. | 1023 |
| 149 | Lahtinen, Rev. William | Rev. | 1041 |
| 164 | Peruschitz, Rev. Joseph Maria | Rev. | 1056 |
| 202 | Astor, Col. John Jacob | Col. | 1094 |
| 293 | Dodge, Dr. Washington | Dr. | 1185 |
| 414 | Oliva y Ocana, Dona. Fermina | Dona. | 1306 |

[389]:
```
def TitleTest(x,y):
    if x in ['Mr.', 'Mrs.', 'Miss.', 'Master.']:
        return x
    elif y  == 1306:
        return 'Mrs.'
    elif y  == 980:
        return 'Miss.'
    else:
        return 'Mr.'

test['Title'] = test.apply(lambda x: TitleTest(x['title'], x['PassengerId']),␣
↪axis = 1)
```

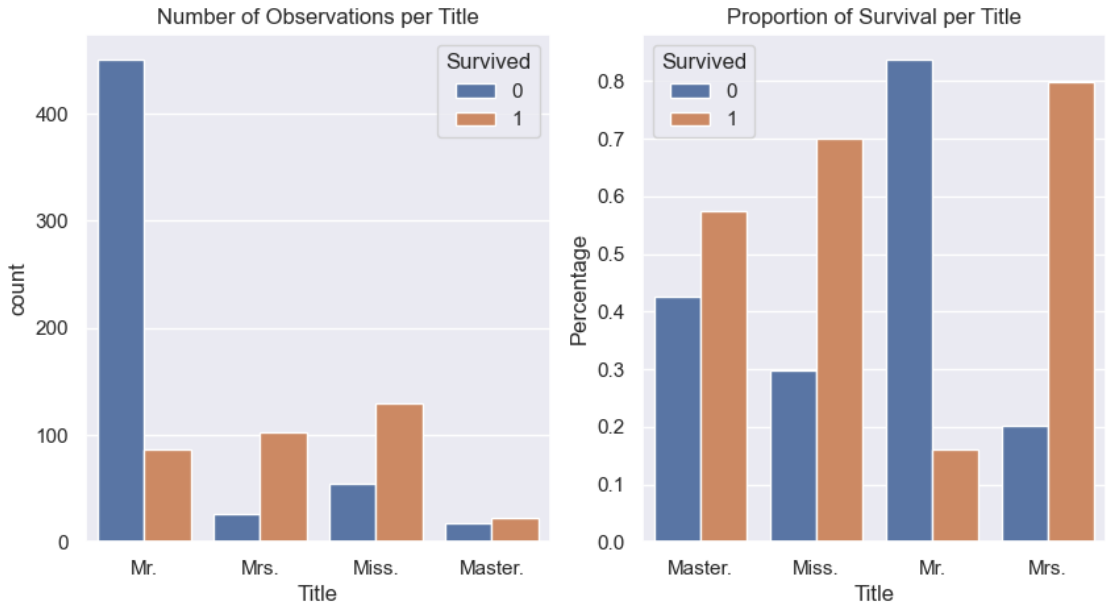
[390]:
```
def lastName(x):
    l = str.split(x)
    for j in range(len(l)):
        if l[j][-1] == ',':
            if j == 0:
                return l[j]
            elif j == 1:
                return l[0] + l[1]
            elif j == 2:
                return l[0] + l[1] + l[2]
            else:
                print('Wrong Parsing')

train['LastName'] = train.apply(lambda x: lastName(x['Name']), axis = 1)
test['LastName'] = test.apply(lambda x: lastName(x['Name']), axis = 1)

train.drop(['Name', 'title', 'PassengerId'], axis = 1, inplace = True)
test.drop(['Name', 'title', 'PassengerId'], axis = 1, inplace = True)
```

[392]:
```
# First, compute aggregated stats for each family in train:
family_stats = train.groupby('LastName')['Survived'].agg(
    total_survived='sum',
```

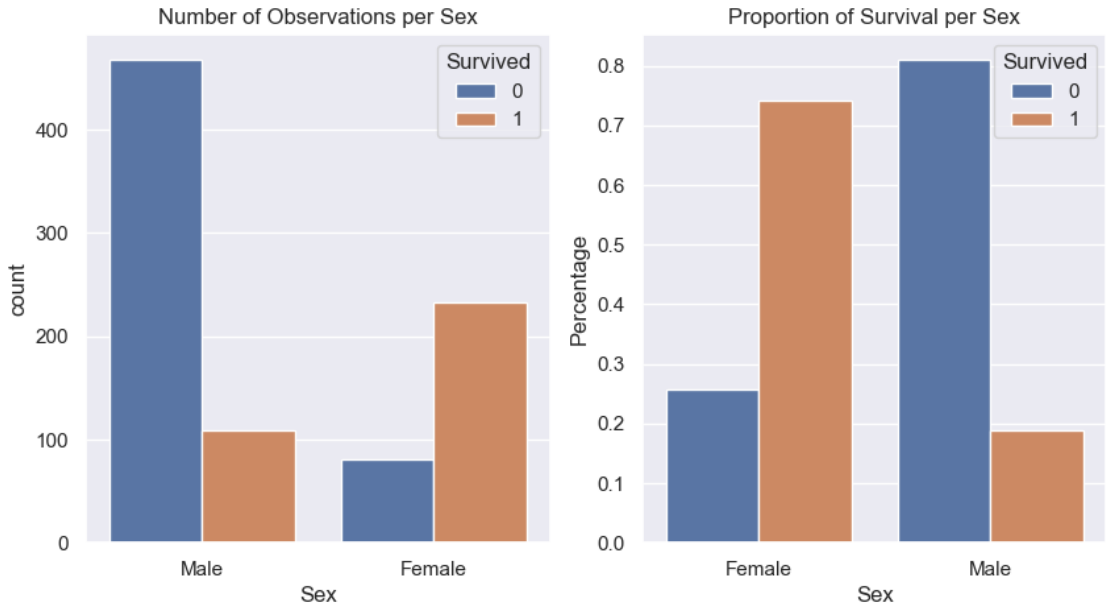
```
    family_count='count'
)

# Define a function for the train rows. For each row, we exclude itself.
def assign_train_outcome(row):
    stats = family_stats.loc[row['LastName']]
    # No other family members available
    if (stats['family_count'] == 1) or (row['Parch'] + row['SibSp'] == 0):
        return 1
    stats = family_stats.loc[row['LastName']]
    # Compute survivors among other members (excluding self)
    others_survived = stats['total_survived'] - row['Survived']
    if others_survived > 0:
        return 2
    else:
        return 0

# Apply the function to each row in train
train['FamilyOutcome'] = train.apply(assign_train_outcome, axis=1)

# For the test dataframe, we have no individual survival info.
# We simply use the aggregated family stats from train.
def assign_test_outcome(family, P, S):
    # If the family is not seen in train, we have no information.
    if (family not in family_stats.index) or ((P + S) == 0):
        return 1
    stats = family_stats.loc[family]
    # If any passenger in train (for this family) survived, assume family␣
 ↪survival.
    if stats['total_survived'] > 0:
        return 2
    else:
        return 0

# Apply to test data
test['FamilyOutcome'] = test.apply(lambda x :␣
 ↪assign_test_outcome(x['LastName'], x['Parch'], x['SibSp']), axis = 1)

train.drop('LastName', axis = 1, inplace = True)
test.drop('LastName', axis = 1, inplace = True)
```

[401]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 5))
sns.countplot(x = 'Title', data = train, hue = 'Survived', ax = axes[0])
axes[0].set_title('Number of Observations per Title')

survival_rates = train.groupby('Title')["Survived"].value_counts(normalize =␣
 ↪True).unstack() ### Group and normalize for percentage
```

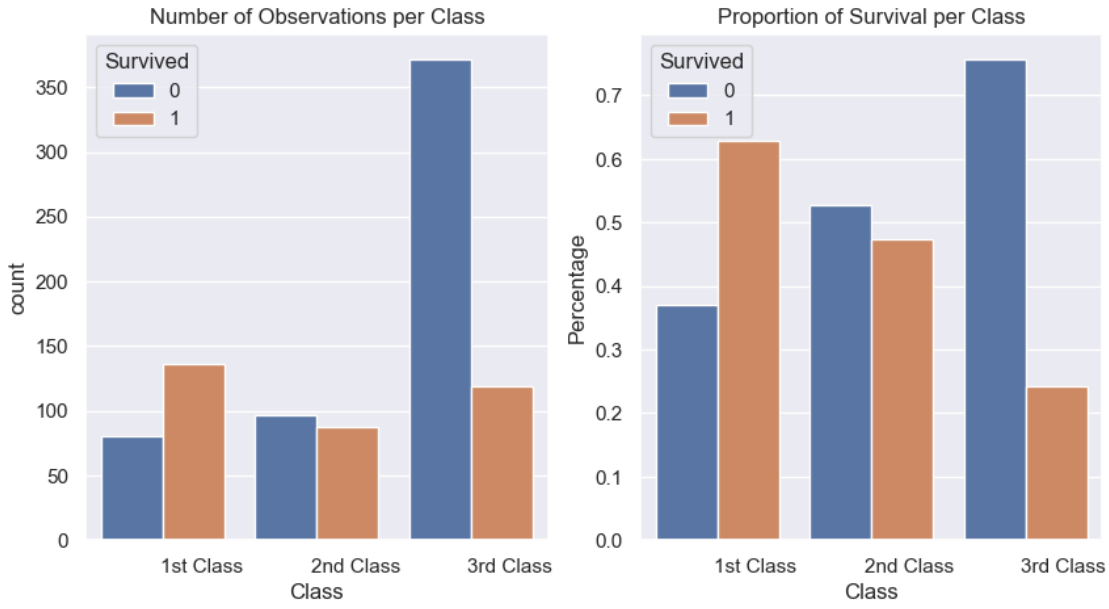
```
# Reset index and melt into long format for seaborn
survival_long = survival_rates.reset_index().melt(id_vars='Title',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='Title', y="Percentage", hue="Survived", ax =␣
 ↪axes[1])
axes[1].set_title('Proportion of Survival per Title');
```




[402]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 5))
sns.countplot(x = 'Sex', data = train, hue = 'Survived', ax = axes[0])
axes[0].set_title('Number of Observations per Sex')

survival_rates = train.groupby('Sex')["Survived"].value_counts(normalize =␣
 ↪True).unstack() ### Group and normalize for percentage
# Reset index and melt into long format for seaborn
survival_long = survival_rates.reset_index().melt(id_vars='Sex',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='Sex', y="Percentage", hue="Survived", ax =␣
 ↪axes[1])
axes[1].set_title('Proportion of Survival per Sex');
```




[403]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 5))
sns.countplot(x = 'Pclass', data = train, hue = 'Survived', ax = axes[0])
axes[0].set_title('Number of Observations per Class')
x = np.arange(len(pd.unique(train.Pclass)))
width = 0.25
axes[0].set_xlabel('Class')
axes[0].set_xticks(ticks =x + width, labels = ['1st Class', '2nd Class', '3rd␣
 ↪Class']);

survival_rates = train.groupby('Pclass')["Survived"].value_counts(normalize =␣
 ↪True).unstack() ### Group and normalize for percentage
# Reset index and melt into long format for seaborn
survival_long = survival_rates.reset_index().melt(id_vars='Pclass',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='Pclass', y="Percentage", hue="Survived", ax␣
 ↪= axes[1])
axes[1].set_title('Proportion of Survival per Class');
axes[1].set_xlabel('Class')
axes[1].set_xticks(ticks =x + width, labels = ['1st Class', '2nd Class', '3rd␣
 ↪Class']);
```




[404]:
```
import itertools
from scipy.stats import chi2_contingency

def best_segmentation(train_df, test_df, column, n_segments, n_candidates,␣
 ↪test):
    """
    Finds the best segmentation (binning) of a continuous variable into␣
 ↪n_segments bins
    based on minimizing the chi-squared test p-value for independence with the␣
 ↪"Survived" column,
    and then adds the binned column as a new column (named column+'New') to␣
 ↪both the training and test dataframes.

    Parameters:
      train_df    : pandas.DataFrame
                    The training dataframe containing the data.
      test_df     : pandas.DataFrame
                    The test dataframe that will receive the segmented version␣
 ↪of the column.
      column      : str
                    The name of the continuous column to segment.
      n_segments  : int
                    The desired number of bins.
      n_candidates: int
                    The number of evenly spaced candidate endpoints between the␣
 ↪min and max of train_df[column].
```

```
    Returns:
      None. The function modifies train_df and test_df in place.
    """
    # Generate candidate endpoints from the training data.
    col_data = train_df[column]
    #candidates = np.linspace(col_data.min(), col_data.max(), n_candidates)

    best_p = 1.0
    best_endpoints = None

    quants = np.quantile(a = col_data, q = [j / n_candidates for j in range(1,␣
↪n_candidates)])
    # To create n_segments bins, we need n_segments - 1 endpoints.
    #for endpoints in itertools.combinations(candidates, n_segments - 1):\
    for endpoints in itertools.combinations(quants, n_segments - 1):
        #endpoints = sorted(np.random.choice(col_data, size = n_segments - 1,␣
↪replace=False))
        # np.digitize assigns bins as follows:
        #   Bin 0: values < endpoints[0]
        #   Bin i: endpoints[i-1] <= value < endpoints[i] for i=1,...
↪,len(endpoints)
        #   Bin len(endpoints): values >= endpoints[-1]
        segmentation = np.digitize(col_data, endpoints, right=False)
        if test == 'Chi2':
             # Build the contingency table for the chi-squared test.
            contingency_table = pd.crosstab(segmentation, train_df['Survived'])
            _, p, _, _ = chi2_contingency(contingency_table)
        elif test == 'Fisher':
            p = 0
            for i in range(n_segments - 1):
                contingency_table = pd.crosstab(((segmentation == i) |␣
↪(segmentation == i + 1)).astype('int'), train_df['Survived'])
                if contingency_table.shape != (2,2):
                    continue
                _, p1 = fisher_exact(contingency_table)
                p = p + p1
        if p < best_p:
            best_p = p
            best_endpoints = endpoints
    if best_endpoints is None:
        raise ValueError("No valid endpoints found. Check your parameters.")

    # Add the new segmentation column to both dataframes.
    new_col = column + 'New'
    trainCol = np.digitize(train_df[column], best_endpoints, right=False)
    testCol = np.digitize(test_df[column], best_endpoints, right=False)
```

```
    return best_endpoints, best_p, trainCol, testCol
```

[405]:
```
train1stClass = train.loc[train.Pclass == 1, :]
test1stClass = test.loc[test.Pclass == 1, :]
train2ndClass = train.loc[train.Pclass == 2, :]
test2ndClass = test.loc[test.Pclass == 2, :]
train3rdClass = train.loc[train.Pclass == 3, :]
test3rdClass = test.loc[test.Pclass == 3, :]
```

[406]:
```
endpoints1, p1, trainCol1, testCol1 = best_segmentation(train1stClass,␣
 ↪test1stClass, 'Fare', 2, 50, test = 'Chi2')
endpoints2, p2, trainCol2, testCol2 = best_segmentation(train2ndClass,␣
 ↪test2ndClass, 'Fare', 2, 50, test = 'Chi2')
endpoints3, p3, trainCol3, testCol3 = best_segmentation(train3rdClass,␣
 ↪test3rdClass, 'Fare', 2, 50, test = 'Chi2')
```

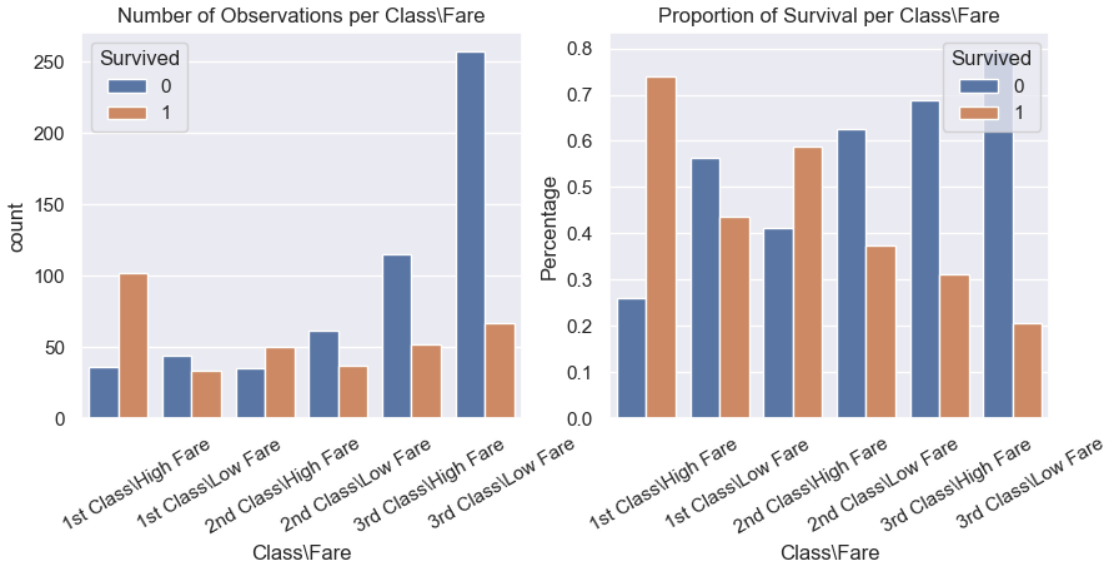
[407]:
```
endpoints2
```

[407]:
```
(18.255129123376644,)
```

[408]:
```
def classFare(x ,y):
    if x == 1:
        if y > endpoints1[0]:
            return '1st Class\High Fare'
        else:
            return '1st Class\Low Fare'
    elif x == 2:
        if y > endpoints2[0]:
            return '2nd Class\High Fare'
        else:
            return '2nd Class\Low Fare'
    else:
        if y > endpoints3[0]:
            return '3rd Class\High Fare'
        else:
            return '3rd Class\Low Fare'

train['ClassFare'] = train.apply(lambda x: classFare(x['Pclass'], x['Fare']),␣
 ↪axis = 1)
test['ClassFare'] = test.apply(lambda x: classFare(x['Pclass'], x['Fare']),␣
 ↪axis = 1)
```

[409]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 4))
labels = ['1st Class\High Fare', '1st Class\Low Fare', '2nd Class\High Fare',
          '2nd Class\Low Fare', '3rd Class\High Fare', '3rd Class\Low Fare']
sns.countplot(x = 'ClassFare', data = train, hue = 'Survived', ax = axes[0],
              order = labels)
```

```
axes[0].set_title('Number of Observations per Class\Fare')
x = np.arange(len(pd.unique(train.ClassFare)))
width = 0.25
axes[0].set_xlabel('Class\Fare')
axes[0].set_xticks(ticks =x + width, labels = labels, rotation = 30);

survival_rates = train.groupby('ClassFare')["Survived"].value_counts(normalize␣
 ↪= True).unstack() ### Group and normalize for percentage
# Reset index and melt into long format for seaborn
survival_long = survival_rates.reset_index().melt(id_vars='ClassFare',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='ClassFare', y="Percentage", hue="Survived",␣
 ↪ax = axes[1])
axes[1].set_title('Proportion of Survival per Class\Fare');
axes[1].set_xlabel('Class\Fare')
axes[1].set_xticks(ticks =x + width, labels = labels, rotation = 30);
```




[410]:
```
# Function to compute survival percentages per FareGroup
def compute_survival_absolute(df, to_split, to_group):
    levels = np.sort(pd.unique(df[to_split]))
    n_plots = len(levels)
    k = len(pd.unique(df[to_group]))
    nRows = int(np.ceil(len(levels) / 2))
    nCols = 2
    if nRows > 1 and n_plots % 2 == 1:
        fig, axes = plt.subplots(nRows - 1, nCols, figsize=(10, 3 * nRows))
    else:
```

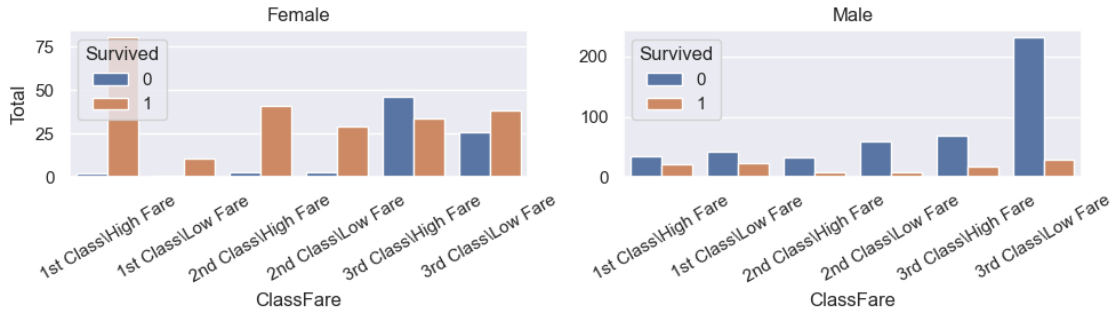
```
        fig, axes = plt.subplots(nRows, nCols, figsize=(10, 3 * nRows))
    for j in range(len(levels)):
        data = df.loc[df[to_split] == levels[j]] ### get subsample
        if n_plots == 2 or n_plots == 3:
            if ((n_plots == 3) and (j < 2)) or n_plots == 2:
                sns.countplot(data=data, x=to_group, hue="Survived", ax=axes[j␣
↪% 2], order = np.sort(pd.unique(df[to_group])))
                axes[j % 2].set_xticks(np.arange(k))
                axes[j % 2].set_title(levels[j % 2])
                if j % 2 == 0:
                    axes[j % 2].set_ylabel("Total")
                else:
                    axes[j % 2].set_ylabel("")
                axes[j % 2].set_xticklabels(axes[j % 2].get_xticklabels(),␣
↪rotation = 30)
            else:
                fig.subplots_adjust(bottom= 2 / 3)
                ax = fig.add_axes([0.2, 0.15, 0.6, 1 / (nRows + 1)])
                sns.countplot(data=data, x=to_group, hue="Survived", order = np.
↪sort(pd.unique(df[to_group])))
                ax.set_title(levels[j])
                ax.set_xlabel(to_group)
                ax.set_xticks(np.arange(k))
                ax.set_xticklabels(ax.get_xticklabels(), rotation = 30)

        else:
            if (n_plots % 2 == 1) and (j ==  2 * nRows - 2):
                plt.tight_layout()
                fig.subplots_adjust(bottom= 1 - 2 / (nRows + 1))
                ax = fig.add_axes([0.2, 0.15, 0.6, 1 / (nRows + 1)])
                sns.countplot(data=data, x=to_group, hue="Survived", order = np.
↪sort(pd.unique(df[to_group])))
                ax.set_title(levels[j])
                ax.set_xlabel(to_group)
                ax.set_xticks(np.arange(k))
                ax.set_xticklabels(ax.get_xticklabels(), rotation = 30)
            else:
                sns.countplot(data=data, x=to_group, hue="Survived", ax=axes[j /
↪/ 2, j % 2], order = np.sort(pd.unique(df[to_group])))
                axes[j // 2, j % 2].set_xticks(np.arange(k))
                axes[j // 2, j % 2].set_title(levels[j])
                axes[j // 2, j % 2].set_xlabel(to_group)
                if j % 2 == 0:
                    axes[j // 2, j % 2].set_ylabel("Total")
                else:
                    axes[j // 2, j % 2].set_ylabel("")
```

```
                axes[j // 2, j % 2].set_xticklabels(axes[j // 2, j % 2].
↪get_xticklabels(), rotation = 30)
        if n_plots % 2 == 0:
            plt.tight_layout()
```

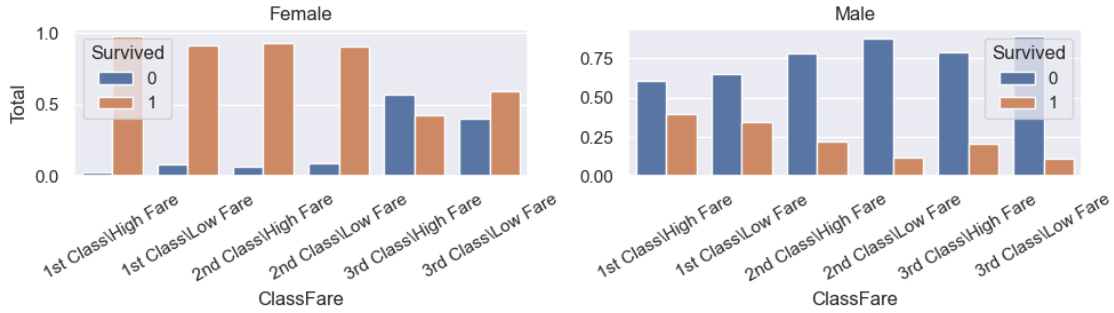
[411]:
```
# Function to compute survival percentages per FareGroup
def compute_survival_percentage(df, to_split, to_group):
    """
    A function to plot the survival proportion inside different levels of a␣
↪categorical levels,
    for subpopulations of another categorical variable.

    Parameters:
        df        : dataframe
        to_split  : Str
                    The categorical column we want to split the data for␣
↪multiple plots
        to_split  : Str
                    The categorical column we want to split the data inside␣
↪each plot
    """
    levels = np.sort(pd.unique(df[to_split]))
    n_plots = len(levels)
    k = len(pd.unique(df[to_group]))
    nRows = int(np.ceil(len(levels) / 2))
    nCols = 2
    if nRows > 1 and n_plots % 2 == 1:
        fig, axes = plt.subplots(nRows - 1, nCols, figsize=(10, 3 * nRows))
    else:
        fig, axes = plt.subplots(nRows, nCols, figsize=(10, 3 * nRows))
    for j in range(len(levels)):
        data = df.loc[df[to_split] == levels[j]] ### get subsample
        survival_rates = data.groupby(to_group)["Survived"].
↪value_counts(normalize = True).unstack() ### Group and normalize for␣
↪percentage
        # Reset index and melt into long format for seaborn
        survival_long = survival_rates.reset_index().melt(id_vars=to_group,␣
↪value_name="Percentage")
        if n_plots == 2 or n_plots == 3:
            if ((n_plots == 3) and (j < 2)) or n_plots == 2:
                sns.barplot(data=survival_long, x=to_group, y="Percentage",␣
↪hue="Survived", ax=axes[j % 2],
                            order = np.sort(pd.unique(df[to_group])))
                axes[j % 2].set_xticks(np.arange(k))
                axes[j % 2].set_title(levels[j % 2])
                if j % 2 == 0:
```

```
                    axes[j % 2].set_ylabel("Total")
                else:
                    axes[j % 2].set_ylabel("")
                axes[j % 2].set_xticklabels(axes[j % 2].get_xticklabels(),␣
↪rotation = 30)
            else:
                fig.subplots_adjust(bottom= 2 / 3)
                ax = fig.add_axes([0.2, 0.15, 0.6, 1 / (nRows + 1)])
                sns.barplot(data=survival_long, x=to_group, y="Percentage",␣
↪hue="Survived",
                            order = np.sort(pd.unique(df[to_group])))
                ax.set_title(levels[j])
                ax.set_xlabel(to_group)
                ax.set_xticks(np.arange(k))
                ax.set_xticklabels(ax.get_xticklabels(), rotation = 30)
        else:
            if (n_plots % 2 == 1) and (j ==  2 * nRows - 2):
                fig.subplots_adjust(bottom= 1 - 2 / (nRows + 1))
                ax = fig.add_axes([0.2, 0.15, 0.6, 1 / (nRows + 1)])
                sns.barplot(data=survival_long, x=to_group, y="Percentage",␣
↪hue="Survived",
                             order = np.sort(pd.unique(df[to_group])))
                ax.set_title(levels[j])
                ax.set_xlabel(to_group)
                ax.set_xticks(np.arange(k))
                ax.set_xticklabels(ax.get_xticklabels(), rotation = 30)
            else:
                sns.barplot(data=survival_long, x=to_group, y="Percentage",␣
↪hue="Survived",
                            ax=axes[j // 2, j % 2], order = np.sort(pd.
↪unique(df[to_group])))
                axes[j // 2, j % 2].set_xticks(np.arange(k))
                axes[j // 2, j % 2].set_title(levels[j])
                axes[j // 2, j % 2].set_xlabel(to_group)
                if j % 2 == 0:
                    axes[j // 2, j % 2].set_ylabel("Total")
                else:
                    axes[j // 2, j % 2].set_ylabel("")
                axes[j // 2, j % 2].set_xticklabels(axes[j // 2, j % 2].
↪get_xticklabels(), rotation = 30)
        if n_plots % 2 == 0:
            plt.tight_layout()
```

[412]:
```
compute_survival_absolute(train, to_split='Sex', to_group='ClassFare')
```




[413]:
```
compute_survival_percentage(train, to_split='Sex', to_group='ClassFare')
```




[414]:
```
train1stClass = train.loc[train.Pclass == 1, :]
test1stClass = test.loc[test.Pclass == 1, :]
train2ndClass = train.loc[train.Pclass == 2, :]
test2ndClass = test.loc[test.Pclass == 2, :]
train3rdClass = train.loc[train.Pclass == 3, :]
test3rdClass = test.loc[test.Pclass == 3, :]

def chi2Subpopulations(df, to_split, to_test, sep):
    p_vals = []
    levels = sorted(pd.unique(df[to_split]))
    for lev in levels:
        subDf = df.loc[df[to_split] == lev, :]
        newCol = (subDf[to_test] >= sep).astype('int')
        newSurv = subDf['Survived']
        contingency_table = pd.crosstab(newCol, newSurv)
        _, p, _, _ = chi2_contingency(contingency_table)
        p_vals.append((lev,p))
    return(p_vals)
res1 = chi2Subpopulations(train1stClass, to_split='Sex', to_test='Fare', sep =␣
 ↪endpoints1[0])
```

```
res2 = chi2Subpopulations(train2ndClass, to_split='Sex', to_test='Fare', sep =␣
 ↪endpoints1[0])
res3 = chi2Subpopulations(train3rdClass, to_split='Sex', to_test='Fare', sep =␣
 ↪endpoints1[0])
print("1st Class", res1[0][0], 'P-value:', res1[0][1])
print("1st Class", res1[1][0], 'P-value:', res1[1][1])
print("2nd Class", res2[0][0], 'P-value:', res2[0][1])
print("2nd Class", res2[1][0], 'P-value:', res2[1][1])
print("3rd Class", res3[0][0], 'P-value:', res3[0][1])
print("3rd Class", res3[1][0], 'P-value:', res3[1][1])
```

```
1st Class Female P-value: 0.8369721745449503
1st Class Male P-value: 0.7505557888383733
2nd Class Female P-value: 1.0
2nd Class Male P-value: 0.7181506464182896
3rd Class Female P-value: 0.2432427057426535
3rd Class Male P-value: 0.007033768473790813
```

[415]:
```
d = {'1st Class\High Fare' : '1st Class', '1st Class\Low Fare' : '1st Class',
    '2nd Class\High Fare' : '2nd Class', '2nd Class\Low Fare' : '2nd Class',
    '3rd Class\High Fare' : '3rd Class', '3rd Class\Low Fare' : '3rd Class'}

train['Class'] = train.ClassFare.map(d)
test['Class'] = test.ClassFare.map(d)

train.drop(['ClassFare', 'Fare', 'Pclass'], inplace = True, axis = 1)
test.drop(['ClassFare', 'Fare', 'Pclass'], inplace = True, axis = 1)
```

[416]:
```
endpoints, p, trainCol, testCol = best_segmentation(train, test, 'Age', 4, 20,␣
 ↪test = 'Fisher')
```

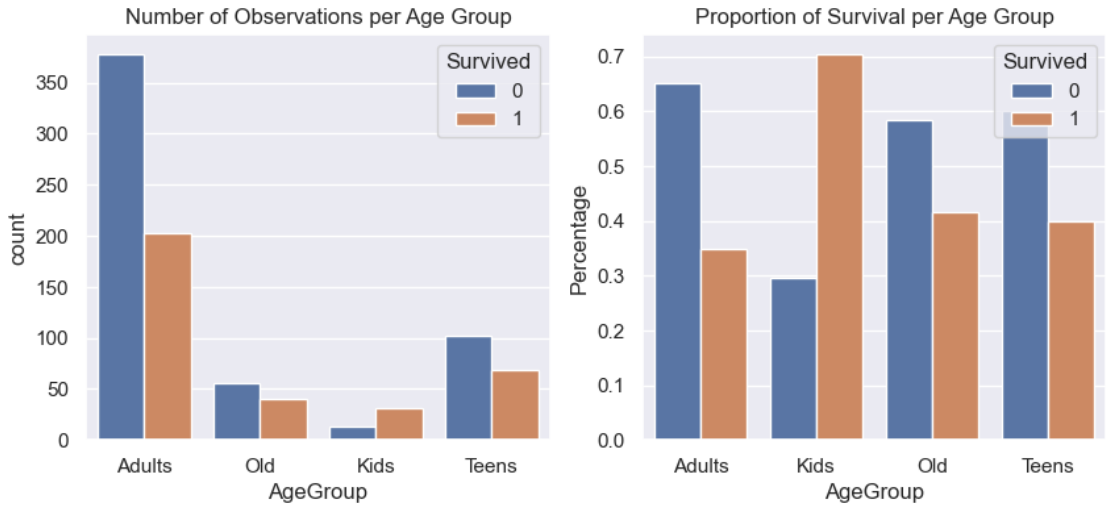
[417]:
```
endpoints
```

[417]:
```
(6.0, 21.0, 48.0)
```

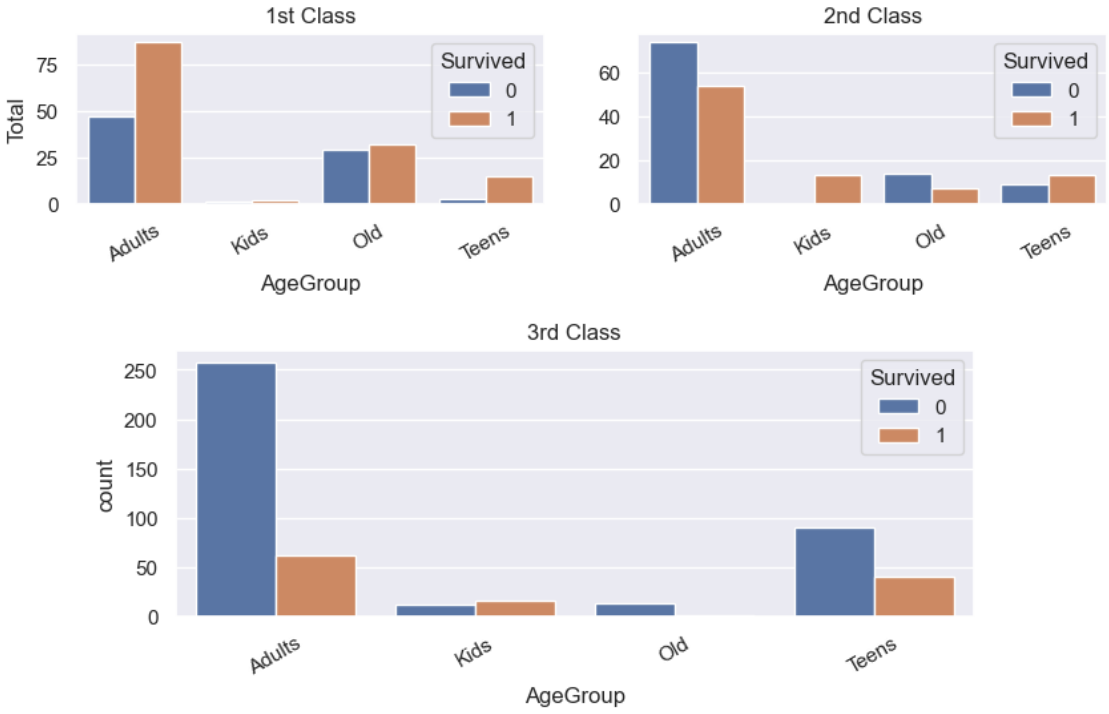
[418]:
```
d = {0: 'Kids', 1: 'Teens', 2: 'Adults', 3: 'Old'}
train['AgeGroup'] = pd.Series(trainCol).map(d)
test['AgeGroup'] = pd.Series(testCol).map(d)

#train.drop('Age', axis = 1, inplace = True)
#test.drop('Age', axis = 1, inplace = True)
```

[419]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 4))
sns.countplot(x = 'AgeGroup', data = train, hue = 'Survived', ax = axes[0])
axes[0].set_title('Number of Observations per Age Group')

survival_rates = train.groupby('AgeGroup')["Survived"].value_counts(normalize =␣
 ↪True).unstack() ### Group and normalize for percentage
```

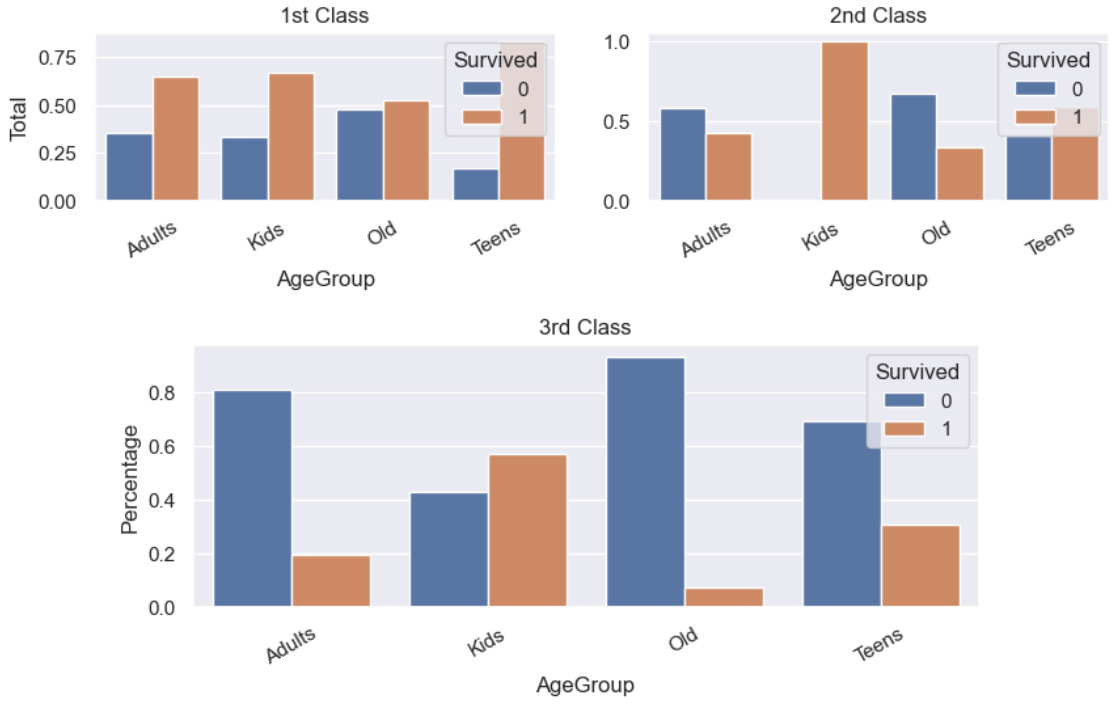
```
# Reset index and melt into long format for seaborn
survival_long = survival_rates.reset_index().melt(id_vars='AgeGroup',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='AgeGroup', y="Percentage", hue="Survived",␣
 ↪ax = axes[1])
axes[1].set_title('Proportion of Survival per Age Group');
```



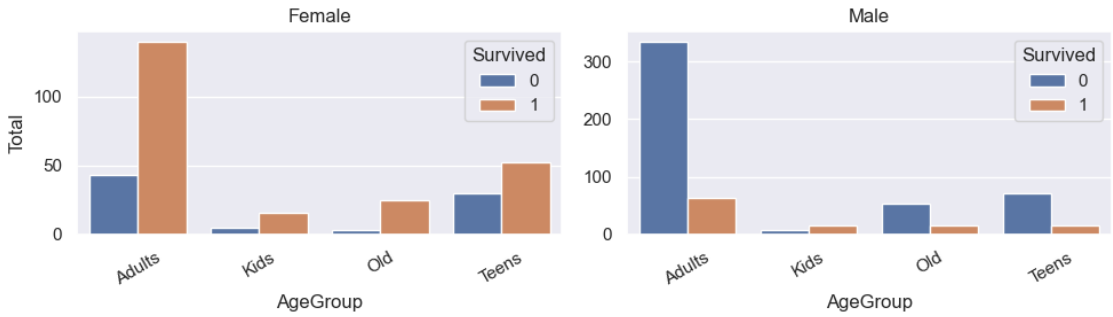

[420]:
```
compute_survival_absolute(train, to_split='Class', to_group='AgeGroup')
```



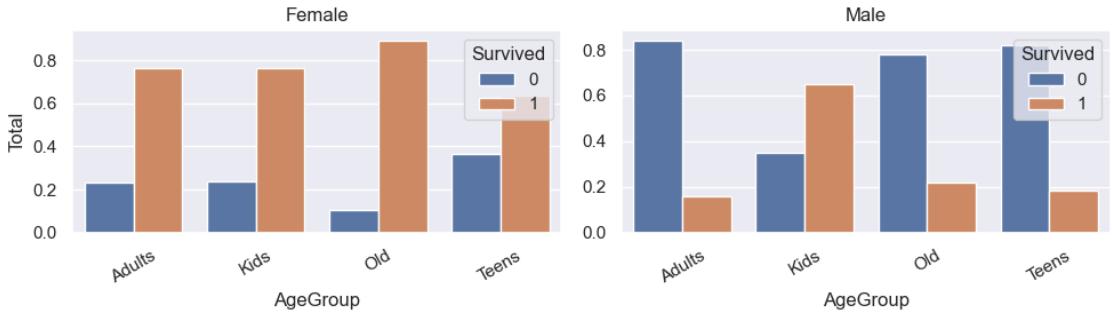
[421]:
```
compute_survival_percentage(train, to_split='Class', to_group='AgeGroup')
```




[422]:
```
compute_survival_absolute(train, to_split='Sex', to_group='AgeGroup')
```




[423]:
```
compute_survival_percentage(train, to_split='Sex', to_group='AgeGroup')
```




[424]:
```
import statsmodels.api as sm

# Prepare the data
X = sm.add_constant(train['Age'])  # add intercept
y = train['Survived']

# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.662244
         Iterations 4
                           Logit Regression Results
==============================================================================
Dep. Variable:               Survived   No. Observations:                  891
Model:                          Logit   Df Residuals:                      889
Method:                           MLE   Df Model:                            1
Date:                Fri, 28 Feb 2025   Pseudo R-squ.:                0.005509
Time:                        06:43:37   Log-Likelihood:                -590.06
converged:                       True   LL-Null:                       -593.33
Covariance Type:            nonrobust   LLR p-value:                   0.01057
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0950      0.163     -0.582      0.560      -0.415       0.225
Age           -0.0129      0.005     -2.535      0.011      -0.023      -0.003
==============================================================================
```

[425]:
```
# Prepare the data
X = sm.add_constant(train.loc[train.Sex == 'Male', 'Age'])  # add intercept
y = train.loc[train.Sex == 'Male', 'Survived']
```

```
# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.479594
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                Survived   No. Observations:                  577
Model:                           Logit   Df Residuals:                      575
Method:                            MLE   Df Model:                            1
Date:                 Fri, 28 Feb 2025   Pseudo R-squ.:                 0.01040
Time:                         06:43:38   Log-Likelihood:                -276.73
converged:                        True   LL-Null:                       -279.63
Covariance Type:            nonrobust   LLR p-value:                   0.01586
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.8824      0.257     -3.431      0.001      -1.386      -0.378
Age           -0.0191      0.008     -2.368      0.018      -0.035      -0.003
==============================================================================
```

[426]:
```
# Prepare the data
X = sm.add_constant(train.loc[train.Sex == 'Female', 'Age'])  # add intercept
y = train.loc[train.Sex == 'Female', 'Survived']

# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.558875
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:                Survived   No. Observations:                  314
Model:                           Logit   Df Residuals:                      312
Method:                            MLE   Df Model:                            1
Date:                 Fri, 28 Feb 2025   Pseudo R-squ.:                 0.02109
Time:                         06:43:38   Log-Likelihood:                -175.49
```

```
converged:                       True   LL-Null:                       -179.27
Covariance Type:            nonrobust   LLR p-value:                  0.005965
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.3495      0.284      1.229      0.219      -0.208       0.907
Age            0.0275      0.010      2.673      0.008       0.007       0.048
==============================================================================
```

[427]:
```
# Prepare the data
X = sm.add_constant(train1stClass.loc[:, 'Age'])  # add intercept
y = train1stClass.loc[:, 'Survived']

# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.621762
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:               Survived   No. Observations:                  216
Model:                          Logit   Df Residuals:                      214
Method:                           MLE   Df Model:                            1
Date:                Fri, 28 Feb 2025   Pseudo R-squ.:                 0.05672
Time:                        06:43:39   Log-Likelihood:                -134.30
converged:                       True   LL-Null:                       -142.38
Covariance Type:            nonrobust   LLR p-value:                 5.844e-05
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.2082      0.472      4.682      0.000       1.284       3.133
Age           -0.0424      0.011     -3.824      0.000      -0.064      -0.021
==============================================================================
```

[428]:
```
# Prepare the data
X = sm.add_constant(train2ndClass.loc[:, 'Age'])  # add intercept
y = train2ndClass.loc[:, 'Survived']

# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
```

```
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.645125
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:                Survived   No. Observations:                  184
Model:                           Logit   Df Residuals:                      182
Method:                            MLE   Df Model:                            1
Date:                 Fri, 28 Feb 2025   Pseudo R-squ.:                 0.06729
Time:                         06:43:39   Log-Likelihood:                -118.70
converged:                        True   LL-Null:                       -127.27
Covariance Type:            nonrobust   LLR p-value:                 3.494e-05
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.3216      0.400      3.302      0.001       0.537       2.106
Age           -0.0474      0.012     -3.845      0.000      -0.072      -0.023
==============================================================================
```

[429]:
```
# Prepare the data
X = sm.add_constant(train3rdClass.loc[:, 'Age'])  # add intercept
y = train3rdClass.loc[:, 'Survived']

# Fit the logistic regression model
model = sm.Logit(y, X)
result = model.fit()

# View the summary with coefficients, standard errors, and confidence intervals
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.530064
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                Survived   No. Observations:                  491
Model:                           Logit   Df Residuals:                      489
Method:                            MLE   Df Model:                            1
Date:                 Fri, 28 Feb 2025   Pseudo R-squ.:                 0.04284
Time:                         06:43:41   Log-Likelihood:                -260.26
converged:                        True   LL-Null:                       -271.91
Covariance Type:            nonrobust   LLR p-value:                 1.388e-06
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0137      0.255     -0.054      0.957      -0.513       0.486
```

```
Age             -0.0472       0.010      -4.594       0.000       -0.067       -0.027
==============================================================================
```

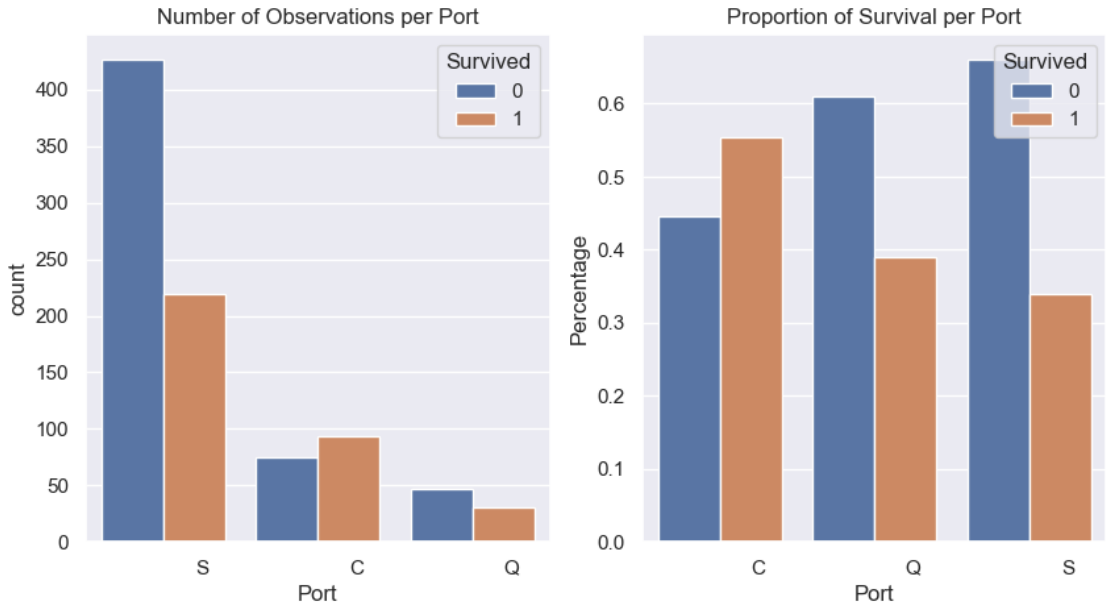
[430]:
```
train.drop('AgeGroup', axis = 1, inplace = True)
test.drop('AgeGroup', axis = 1, inplace = True)
```

[431]:
```
train
```

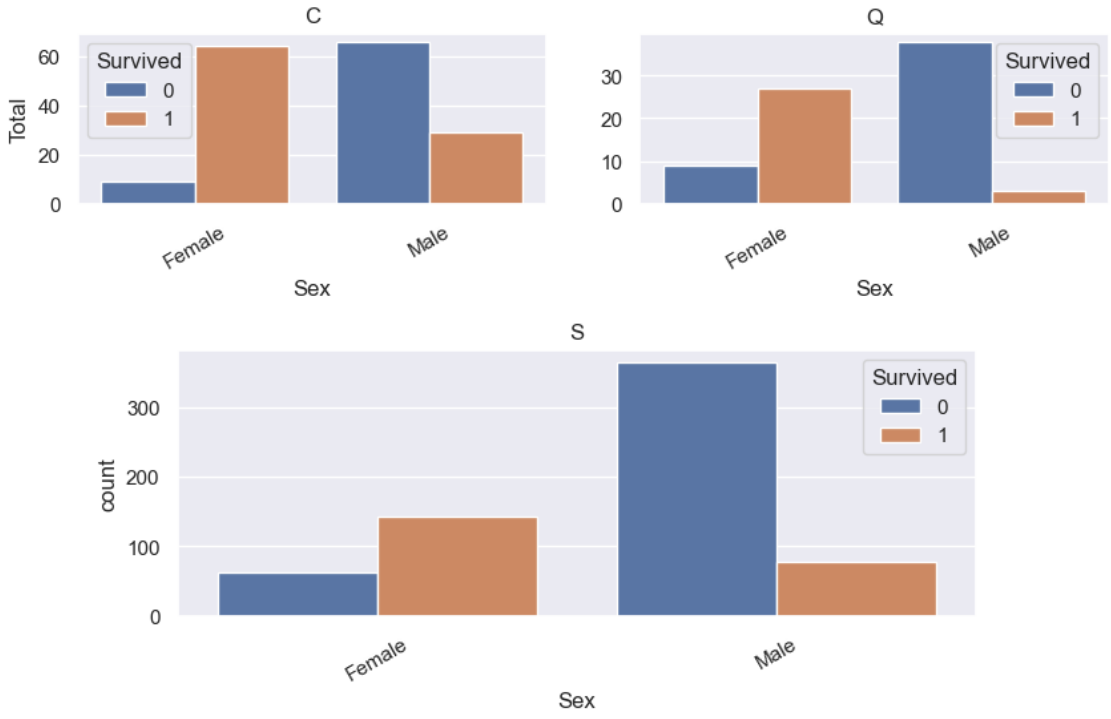
[431]:
```
            Age  SibSp  Parch  Survived Embarked     Sex  Title  FamilyOutcome  \
0     22.000000    1.0    0.0         0        S    Male    Mr.              0
1     38.000000    1.0    0.0         1        C  Female   Mrs.              1
2     26.000000    0.0    0.0         1        S  Female  Miss.              1
3     35.000000    1.0    0.0         1        S  Female   Mrs.              0
4     35.000000    0.0    0.0         0        S    Male    Mr.              1
..          …    …      …       …       …      …     …            …
886   27.000000    0.0    0.0         0        S    Male    Mr.              1
887   19.000000    0.0    0.0         1        S  Female  Miss.              1
888   17.155111    1.0    2.0         0        S  Female  Miss.              0
889   26.000000    0.0    0.0         1        C    Male    Mr.              1
890   32.000000    0.0    0.0         0        Q    Male    Mr.              1

          Class
0     3rd Class
1     1st Class
2     3rd Class
3     1st Class
4     3rd Class
..          …
886   2nd Class
887   1st Class
888   3rd Class
889   1st Class
890   3rd Class

[891 rows x 9 columns]
```

[432]:
```
fix, axes = plt.subplots(1, 2, figsize = (10, 5))
sns.countplot(x = 'Embarked', data = train, hue = 'Survived', ax = axes[0])
axes[0].set_title('Number of Observations per Port')
x = np.arange(len(pd.unique(train.Embarked)))
width = 0.25
axes[0].set_xlabel('Port')
axes[0].set_xticks(ticks =x + width);

survival_rates = train.groupby('Embarked')["Survived"].value_counts(normalize =␣
 ↪True).unstack() ### Group and normalize for percentage
# Reset index and melt into long format for seaborn
```

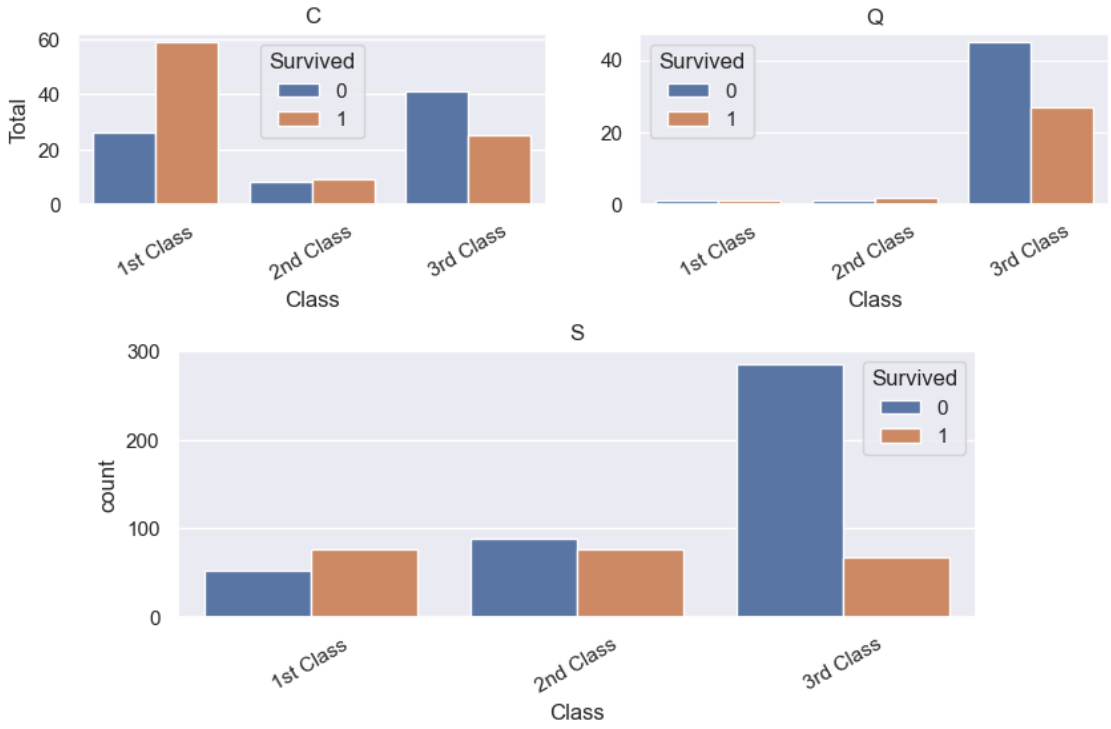
```
survival_long = survival_rates.reset_index().melt(id_vars='Embarked',␣
 ↪value_name="Percentage")
sns.barplot(data=survival_long, x='Embarked', y="Percentage", hue="Survived",␣
 ↪ax = axes[1])
axes[1].set_title('Proportion of Survival per Port');
axes[1].set_xlabel('Port')
axes[1].set_xticks(ticks =x + width);
```




[433]:
```
compute_survival_absolute(train, to_split='Embarked', to_group='Sex')
```




[434]:
```
compute_survival_absolute(train, to_split='Embarked', to_group='Class')
```



[435]:
```
train['ClassSex'] = train.Class + train.Sex
```

[436]:
```
from scipy.stats import fisher_exact

def iterate_pairs(data):
        for i in range(len(data)):
            for j in range(i + 1, len(data)):
                yield data[i], data[j]

for j in pd.unique(train.ClassSex):
    print(j)
    subDf = train.loc[train.ClassSex == j,:]
    for pair in iterate_pairs(pd.unique(subDf.Embarked)):
        print(pair)
        subDf2 = subDf.loc[(train.Embarked == pair[0]) | (train.Embarked ==␣
 ↪pair[1]), :]
        contingency_table = pd.crosstab(subDf2.Embarked, subDf2.Survived)
        if contingency_table.shape != (2,2):
            continue
        print(contingency_table)
        _, p = fisher_exact(contingency_table)
        print(p)
```

```
3rd ClassMale
('S', 'Q')
Survived    0   1
Embarked
Q          36   3
S         231  34
0.44306123670540964
('S', 'C')
Survived    0   1
Embarked
C          33  10
S         231  34
0.09651310562994166
('Q', 'C')
Survived   0   1
Embarked
C         33  10
Q         36   3
0.07163085184452506
1st ClassFemale
('C', 'S')
Survived  0   1
Embarked
```

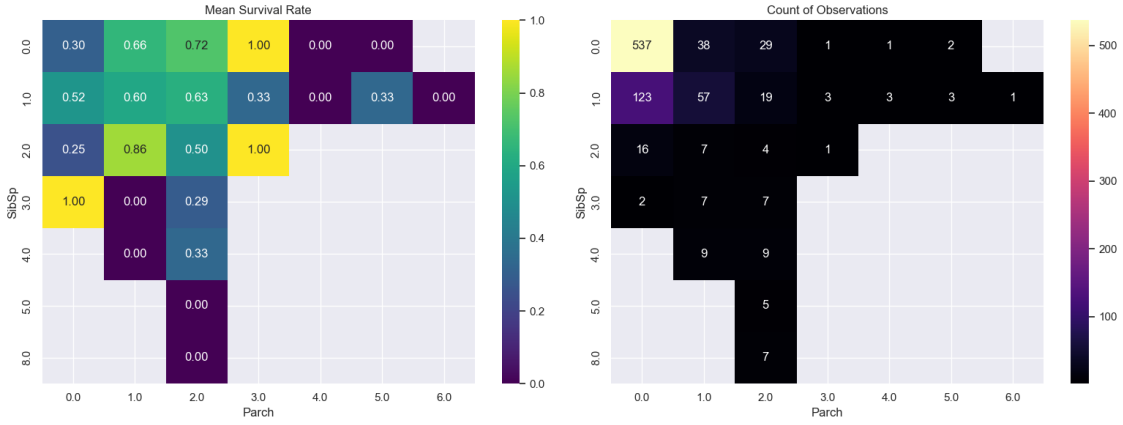
```
C         1  42
S         2  48
1.0
('C', 'Q')
Survived  0   1
Embarked
C         1  42
Q         0   1
1.0
('S', 'Q')
Survived  0   1
Embarked
Q         0   1
S         2  48
1.0
3rd ClassFemale
('S', 'C')
Survived   0   1
Embarked
C          8  15
S         55  33
0.019825694829963252
('S', 'Q')
Survived   0   1
Embarked
Q          9  24
S         55  33
0.000905234478035054
('C', 'Q')
Survived  0   1
Embarked
C         8  15
Q         9  24
0.5690816233357147
1st ClassMale
('S', 'C')
Survived   0   1
Embarked
C         25  17
S         51  28
0.693194984759524
('S', 'Q')
Survived   0   1
Embarked
Q          1   0
S         51  28
1.0
('C', 'Q')
```

```
Survived   0   1
Embarked
C         25  17
Q          1   0
1.0
2nd ClassFemale
('C', 'S')
Survived  0   1
Embarked
C         0   7
S         6  61
1.0
('C', 'Q')
('S', 'Q')
Survived  0   1
Embarked
Q         0   2
S         6  61
1.0
2nd ClassMale
('S', 'C')
Survived   0   1
Embarked
C          8   2
S         82  15
0.658304194091278
('S', 'Q')
Survived   0   1
Embarked
Q          1   0
S         82  15
1.0
('C', 'Q')
Survived  0  1
Embarked
C         8  2
Q         1  0
1.0
```

[437]:
```
train.drop('Embarked', axis = 1, inplace = True)
test.drop('Embarked', axis = 1, inplace = True)
train.drop('ClassSex', axis = 1, inplace = True)
```

[438]:
```
# Calculate the mean survival rate for each combination of SibSp and Parch
mean_survival = train.groupby(['SibSp', 'Parch'])['Survived'].mean().
  ↪reset_index()
```

```
heatmap_mean = mean_survival.pivot(index='SibSp', columns='Parch',␣
 ↪values='Survived')

# Calculate the count of observations for each combination
count_observations = train.groupby(['SibSp', 'Parch']).size().
 ↪reset_index(name='Count')
heatmap_count = count_observations.pivot(index='SibSp', columns='Parch',␣
 ↪values='Count')

# Create subplots to plot both heatmaps side by side
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot the heatmap for the mean survival rate
sns.heatmap(heatmap_mean, annot=True, fmt=".2f", cmap="viridis", ax=axes[0])
axes[0].set_title("Mean Survival Rate")
axes[0].set_xlabel("Parch")
axes[0].set_ylabel("SibSp")

# Plot the heatmap for the count of observations
sns.heatmap(heatmap_count, annot=True, fmt=".0f", cmap="magma", ax=axes[1])
axes[1].set_title("Count of Observations")
axes[1].set_xlabel("Parch")
axes[1].set_ylabel("SibSp")

plt.tight_layout()
plt.show()
```




[439]:
```
def FamilySize(x, y):
    if x == 0 and y == 0:
        return 'Alone'
    elif x <= 2 and y <= 1:
        return 'Small'
```

```
    else:
        return 'Large'

train['FamilySize'] = train.apply(lambda x: FamilySize(x['Parch'], x['SibSp']),␣
 ↪axis = 1)
test['FamilySize'] = test.apply(lambda x: FamilySize(x['Parch'], x['SibSp']),␣
 ↪axis = 1)
```

[440]:
```
train.drop(['Sex', 'Parch','SibSp'], axis = 1, inplace = True)
test.drop(['Sex', 'Parch', 'SibSp'], axis = 1, inplace = True)
```

[441]:
```
test
```

[441]:

| | Age | Title | FamilyOutcome | Class | FamilySize |
|---|---|---|---|---|---|
| 0 | 34.500000 | Mr. | 1 | 3rd Class | Alone |
| 1 | 47.000000 | Mrs. | 1 | 3rd Class | Small |
| 2 | 62.000000 | Mr. | 1 | 2nd Class | Alone |
| 3 | 27.000000 | Mr. | 1 | 3rd Class | Alone |
| 4 | 22.000000 | Mrs. | 2 | 3rd Class | Small |
| .. | … | … | … | … | … |
| 413 | 30.517638 | Mr. | 1 | 3rd Class | Alone |
| 414 | 39.000000 | Mrs. | 1 | 1st Class | Alone |
| 415 | 38.500000 | Mr. | 1 | 3rd Class | Alone |
| 416 | 30.517638 | Mr. | 1 | 3rd Class | Alone |
| 417 | 22.833750 | Master. | 2 | 3rd Class | Small |

[418 rows x 5 columns]

[442]:
```
train
```

[442]:

| | Age | Survived | Title | FamilyOutcome | Class | FamilySize |
|---|---|---|---|---|---|---|
| 0 | 22.000000 | 0 | Mr. | 0 | 3rd Class | Small |
| 1 | 38.000000 | 1 | Mrs. | 1 | 1st Class | Small |
| 2 | 26.000000 | 1 | Miss. | 1 | 3rd Class | Alone |
| 3 | 35.000000 | 1 | Mrs. | 0 | 1st Class | Small |
| 4 | 35.000000 | 0 | Mr. | 1 | 3rd Class | Alone |
| .. | … | … | … | … | … | … |
| 886 | 27.000000 | 0 | Mr. | 1 | 2nd Class | Alone |
| 887 | 19.000000 | 1 | Miss. | 1 | 1st Class | Alone |
| 888 | 17.155111 | 0 | Miss. | 0 | 3rd Class | Small |
| 889 | 26.000000 | 1 | Mr. | 1 | 1st Class | Alone |
| 890 | 32.000000 | 0 | Mr. | 1 | 3rd Class | Alone |

[891 rows x 6 columns]

[443]:
```
d = {'1st Class' : 1, '2nd Class' : 2, '3rd Class' : 3}
```

```
train['Class'] = train.Class.map(d)
test['Class'] = test.Class.map(d)
```

[444]:
```
X_train = train.drop('Survived', axis = 1)
```

[452]:
```
for col in X_train.columns:
    if col == 'FamilySize' or col == 'Title':
        X_train[col] = X_train[col].astype('category')
        test[col] = test[col].astype('category')
    else:
        X_train[col] = X_train[col].astype('float')
        test[col] = test[col].astype('float')
```

[453]:
```
X_train.dtypes
```

[453]:
```
Age              float64
Title           category
FamilyOutcome    float64
Class            float64
FamilySize      category
dtype: object
```

[447]:
```
X_train
```

[447]:

| | Age | Title | FamilyOutcome | Class | FamilySize |
|---|---|---|---|---|---|
| 0 | 22.000000 | Mr. | 0 | 3 | Small |
| 1 | 38.000000 | Mrs. | 1 | 1 | Small |
| 2 | 26.000000 | Miss. | 1 | 3 | Alone |
| 3 | 35.000000 | Mrs. | 0 | 1 | Small |
| 4 | 35.000000 | Mr. | 1 | 3 | Alone |
| .. | … | … | … | … | … |
| 886 | 27.000000 | Mr. | 1 | 2 | Alone |
| 887 | 19.000000 | Miss. | 1 | 1 | Alone |
| 888 | 17.155111 | Miss. | 0 | 3 | Small |
| 889 | 26.000000 | Mr. | 1 | 1 | Alone |
| 890 | 32.000000 | Mr. | 1 | 3 | Alone |

[891 rows x 5 columns]

[448]:
```
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split, GridSearchCV

cat_features = ['Title', 'FamilySize']

# Initialize CatBoostClassifier with verbose disabled for grid search
cat_model = CatBoostClassifier(cat_features=cat_features, verbose=0,␣
 ↪iterations=1200, eval_fraction=0.1, early_stopping_rounds=7)
```

```
# Define the hyperparameter grid
param_grid = {
    'depth': [2, 3, 4],
    'learning_rate': [0.005 * j for j in range(1, 5)],
    'l2_leaf_reg': [0.001 * j for j in range(5)]
}

grid_search = GridSearchCV(
    estimator=cat_model,
    param_grid=param_grid,
    cv=5,              # 5-fold cross-validation
    scoring='accuracy',# You can change the scoring metric if desired
    n_jobs=3
)


grid_search.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
```

```
Best parameters found:  {'depth': 2, 'l2_leaf_reg': 0.0, 'learning_rate': 0.01}
Best cross-validation accuracy: 0.8439708743958321
```

[449]:
```
catBoost = CatBoostClassifier(depth = grid_search.best_params_['depth'],␣
 ↪iterations=1200,
                              learning_rate = grid_search.
 ↪best_params_['learning_rate'], cat_features=cat_features,
                              l2_leaf_reg = grid_search.
 ↪best_params_['l2_leaf_reg'])

X_train_1, X_val, y_train_1, y_val = train_test_split(
    X_train, y_train, test_size=0.1
)

catBoost.fit(X_train_1, y_train_1, early_stopping_rounds=7, eval_set = (X_val,␣
 ↪y_val))
y_pred = catBoost.predict(test)
y_pred = pd.DataFrame({'PassengerId' : test_id, 'Survived' : y_pred})
y_pred.to_csv('SurvivalPredictionsCatBoost.csv', index = False)
```

```
0:      learn: 0.6868919        test: 0.6869994 best: 0.6869994 (0)     total:
9.67ms   remaining: 11.6s
1:      learn: 0.6802076        test: 0.6798483 best: 0.6798483 (1)     total:
25.5ms   remaining: 15.3s
2:      learn: 0.6742290        test: 0.6739833 best: 0.6739833 (2)     total:
```

```
43.3ms   remaining: 17.3s
3:      learn: 0.6684206        test: 0.6682900 best: 0.6682900 (3)     total:
63.9ms   remaining: 19.1s
4:      learn: 0.6627388        test: 0.6627228 best: 0.6627228 (4)     total:
81.2ms   remaining: 19.4s
5:      learn: 0.6566509        test: 0.6562429 best: 0.6562429 (5)     total:
98.4ms   remaining: 19.6s
6:      learn: 0.6507822        test: 0.6500367 best: 0.6500367 (6)     total:
112ms    remaining: 19.1s
7:      learn: 0.6454915        test: 0.6448719 best: 0.6448719 (7)     total:
128ms    remaining: 19.1s
8:      learn: 0.6403285        test: 0.6398346 best: 0.6398346 (8)     total:
142ms    remaining: 18.8s
9:      learn: 0.6352902        test: 0.6349218 best: 0.6349218 (9)     total:
155ms    remaining: 18.4s
10:     learn: 0.6303736        test: 0.6301305 best: 0.6301305 (10)    total:
169ms    remaining: 18.3s
11:     learn: 0.6255760        test: 0.6254579 best: 0.6254579 (11)    total:
183ms    remaining: 18.1s
12:     learn: 0.6208916        test: 0.6214171 best: 0.6214171 (12)    total:
200ms    remaining: 18.3s
13:     learn: 0.6158496        test: 0.6161116 best: 0.6161116 (13)    total:
216ms    remaining: 18.3s
14:     learn: 0.6113837        test: 0.6122790 best: 0.6122790 (14)    total:
235ms    remaining: 18.6s
15:     learn: 0.6066216        test: 0.6073014 best: 0.6073014 (15)    total:
256ms    remaining: 19s
16:     learn: 0.6023349        test: 0.6031400 best: 0.6031400 (16)    total:
270ms    remaining: 18.8s
17:     learn: 0.5981538        test: 0.5990839 best: 0.5990839 (17)    total:
284ms    remaining: 18.6s
18:     learn: 0.5937396        test: 0.5945030 best: 0.5945030 (18)    total:
299ms    remaining: 18.6s
19:     learn: 0.5897377        test: 0.5906274 best: 0.5906274 (19)    total:
317ms    remaining: 18.7s
20:     learn: 0.5855679        test: 0.5863332 best: 0.5863332 (20)    total:
332ms    remaining: 18.7s
21:     learn: 0.5815527        test: 0.5822304 best: 0.5822304 (21)    total:
349ms    remaining: 18.7s
22:     learn: 0.5776850        test: 0.5783096 best: 0.5783096 (22)    total:
365ms    remaining: 18.7s
23:     learn: 0.5739580        test: 0.5745614 best: 0.5745614 (23)    total:
379ms    remaining: 18.5s
24:     learn: 0.5714294        test: 0.5727449 best: 0.5727449 (24)    total:
393ms    remaining: 18.5s
25:     learn: 0.5692021        test: 0.5708722 best: 0.5708722 (25)    total:
407ms    remaining: 18.4s
26:     learn: 0.5657261        test: 0.5674044 best: 0.5674044 (26)    total:
```

```
420ms    remaining: 18.2s
27:     learn: 0.5623744        test: 0.5640873 best: 0.5640873 (27)    total:
432ms    remaining: 18.1s
28:     learn: 0.5590154        test: 0.5608532 best: 0.5608532 (28)    total:
446ms    remaining: 18s
29:     learn: 0.5557352        test: 0.5576964 best: 0.5576964 (29)    total:
459ms    remaining: 17.9s
30:     learn: 0.5525362        test: 0.5546199 best: 0.5546199 (30)    total:
476ms    remaining: 17.9s
31:     learn: 0.5506242        test: 0.5530506 best: 0.5530506 (31)    total:
490ms    remaining: 17.9s
32:     learn: 0.5476225        test: 0.5501071 best: 0.5501071 (32)    total:
501ms    remaining: 17.7s
33:     learn: 0.5447265        test: 0.5472908 best: 0.5472908 (33)    total:
515ms    remaining: 17.6s
34:     learn: 0.5419309        test: 0.5445942 best: 0.5445942 (34)    total:
534ms    remaining: 17.8s
35:     learn: 0.5390214        test: 0.5422340 best: 0.5422340 (35)    total:
550ms    remaining: 17.8s
36:     learn: 0.5361868        test: 0.5399393 best: 0.5399393 (36)    total:
566ms    remaining: 17.8s
37:     learn: 0.5340750        test: 0.5382681 best: 0.5382681 (37)    total:
582ms    remaining: 17.8s
38:     learn: 0.5315248        test: 0.5358263 best: 0.5358263 (38)    total:
594ms    remaining: 17.7s
39:     learn: 0.5288553        test: 0.5336791 best: 0.5336791 (39)    total:
609ms    remaining: 17.7s
40:     learn: 0.5262546        test: 0.5315918 best: 0.5315918 (40)    total:
625ms    remaining: 17.7s
41:     learn: 0.5236853        test: 0.5291215 best: 0.5291215 (41)    total:
649ms    remaining: 17.9s
42:     learn: 0.5213608        test: 0.5269140 best: 0.5269140 (42)    total:
668ms    remaining: 18s
43:     learn: 0.5191142        test: 0.5247978 best: 0.5247978 (43)    total:
684ms    remaining: 18s
44:     learn: 0.5169416        test: 0.5227676 best: 0.5227676 (44)    total:
701ms    remaining: 18s
45:     learn: 0.5145650        test: 0.5204853 best: 0.5204853 (45)    total:
715ms    remaining: 17.9s
46:     learn: 0.5125031        test: 0.5185752 best: 0.5185752 (46)    total:
729ms    remaining: 17.9s
47:     learn: 0.5105069        test: 0.5167404 best: 0.5167404 (47)    total:
746ms    remaining: 17.9s
48:     learn: 0.5092709        test: 0.5158047 best: 0.5158047 (48)    total:
762ms    remaining: 17.9s
49:     learn: 0.5070599        test: 0.5136812 best: 0.5136812 (49)    total:
776ms    remaining: 17.9s
50:     learn: 0.5049039        test: 0.5116117 best: 0.5116117 (50)    total:
```

```
790ms    remaining: 17.8s
51:     learn: 0.5033409        test: 0.5104358 best: 0.5104358 (51)    total:
804ms    remaining: 17.8s
52:     learn: 0.5012699        test: 0.5084459 best: 0.5084459 (52)    total:
819ms    remaining: 17.7s
53:     learn: 0.4997734        test: 0.5073277 best: 0.5073277 (53)    total:
840ms    remaining: 17.8s
54:     learn: 0.4983095        test: 0.5062383 best: 0.5062383 (54)    total:
854ms    remaining: 17.8s
55:     learn: 0.4966015        test: 0.5046852 best: 0.5046852 (55)    total:
867ms    remaining: 17.7s
56:     learn: 0.4955138        test: 0.5039158 best: 0.5039158 (56)    total:
879ms    remaining: 17.6s
57:     learn: 0.4938840        test: 0.5024459 best: 0.5024459 (57)    total:
892ms    remaining: 17.6s
58:     learn: 0.4925335        test: 0.5014579 best: 0.5014579 (58)    total:
907ms    remaining: 17.5s
59:     learn: 0.4909805        test: 0.5000683 best: 0.5000683 (59)    total:
923ms    remaining: 17.5s
60:     learn: 0.4891684        test: 0.4986763 best: 0.4986763 (60)    total:
937ms    remaining: 17.5s
61:     learn: 0.4873881        test: 0.4969544 best: 0.4969544 (61)    total:
949ms    remaining: 17.4s
62:     learn: 0.4856568        test: 0.4956318 best: 0.4956318 (62)    total:
961ms    remaining: 17.3s
63:     learn: 0.4842327        test: 0.4943682 best: 0.4943682 (63)    total:
973ms    remaining: 17.3s
64:     learn: 0.4825581        test: 0.4927476 best: 0.4927476 (64)    total:
986ms    remaining: 17.2s
65:     learn: 0.4812019        test: 0.4915549 best: 0.4915549 (65)    total:
999ms    remaining: 17.2s
66:     learn: 0.4798859        test: 0.4904062 best: 0.4904062 (66)    total:
1.01s    remaining: 17.1s
67:     learn: 0.4786080        test: 0.4892989 best: 0.4892989 (67)    total:
1.02s    remaining: 17.1s
68:     learn: 0.4770458        test: 0.4877846 best: 0.4877846 (68)    total:
1.04s    remaining: 17s
69:     learn: 0.4755222        test: 0.4863084 best: 0.4863084 (69)    total:
1.06s    remaining: 17.2s
70:     learn: 0.4740362        test: 0.4848693 best: 0.4848693 (70)    total:
1.08s    remaining: 17.2s
71:     learn: 0.4728613        test: 0.4838631 best: 0.4838631 (71)    total:
1.1s     remaining: 17.2s
72:     learn: 0.4717197        test: 0.4828925 best: 0.4828925 (72)    total:
1.12s    remaining: 17.3s
73:     learn: 0.4707232        test: 0.4822075 best: 0.4822075 (73)    total:
1.14s    remaining: 17.3s
74:     learn: 0.4699776        test: 0.4818676 best: 0.4818676 (74)    total:
```

```
1.15s    remaining: 17.3s
75:     learn: 0.4690182        test: 0.4812135 best: 0.4812135 (75)    total:
1.17s    remaining: 17.3s
76:     learn: 0.4680794        test: 0.4805770 best: 0.4805770 (76)    total:
1.19s    remaining: 17.3s
77:     learn: 0.4670384        test: 0.4796991 best: 0.4796991 (77)    total:
1.2s     remaining: 17.3s
78:     learn: 0.4660262        test: 0.4788515 best: 0.4788515 (78)    total:
1.22s    remaining: 17.3s
79:     learn: 0.4646987        test: 0.4778682 best: 0.4778682 (79)    total:
1.23s    remaining: 17.3s
80:     learn: 0.4640851        test: 0.4772448 best: 0.4772448 (80)    total:
1.25s    remaining: 17.3s
81:     learn: 0.4633679        test: 0.4766150 best: 0.4766150 (81)    total:
1.26s    remaining: 17.3s
82:     learn: 0.4621161        test: 0.4753885 best: 0.4753885 (82)    total:
1.28s    remaining: 17.2s
83:     learn: 0.4611919        test: 0.4746260 best: 0.4746260 (83)    total:
1.29s    remaining: 17.2s
84:     learn: 0.4599642        test: 0.4737258 best: 0.4737258 (84)    total:
1.31s    remaining: 17.2s
85:     learn: 0.4594469        test: 0.4734410 best: 0.4734410 (85)    total:
1.32s    remaining: 17.1s
86:     learn: 0.4582624        test: 0.4725769 best: 0.4725769 (86)    total:
1.33s    remaining: 17.1s
87:     learn: 0.4573959        test: 0.4718646 best: 0.4718646 (87)    total:
1.34s    remaining: 17s
88:     learn: 0.4564157        test: 0.4709238 best: 0.4709238 (88)    total:
1.36s    remaining: 17s
89:     learn: 0.4552934        test: 0.4701113 best: 0.4701113 (89)    total:
1.37s    remaining: 17s
90:     learn: 0.4542442        test: 0.4690635 best: 0.4690635 (90)    total:
1.39s    remaining: 17s
91:     learn: 0.4533131        test: 0.4684777 best: 0.4684777 (91)    total:
1.42s    remaining: 17.1s
92:     learn: 0.4525201        test: 0.4678311 best: 0.4678311 (92)    total:
1.43s    remaining: 17.1s
93:     learn: 0.4517473        test: 0.4672047 best: 0.4672047 (93)    total:
1.45s    remaining: 17.1s
94:     learn: 0.4509939        test: 0.4665977 best: 0.4665977 (94)    total:
1.48s    remaining: 17.2s
95:     learn: 0.4499834        test: 0.4656017 best: 0.4656017 (95)    total:
1.49s    remaining: 17.2s
96:     learn: 0.4489975        test: 0.4646302 best: 0.4646302 (96)    total:
1.51s    remaining: 17.2s
97:     learn: 0.4481016        test: 0.4639792 best: 0.4639792 (97)    total:
1.52s    remaining: 17.1s
98:     learn: 0.4471929        test: 0.4630672 best: 0.4630672 (98)    total:
```

```
1.54s    remaining: 17.1s
99:     learn: 0.4462512        test: 0.4624033 best: 0.4624033 (99)    total:
1.56s    remaining: 17.2s
100:    learn: 0.4457334        test: 0.4621011 best: 0.4621011 (100)   total:
1.58s    remaining: 17.2s
101:    learn: 0.4448226        test: 0.4614626 best: 0.4614626 (101)   total:
1.6s     remaining: 17.2s
102:    learn: 0.4439341        test: 0.4608426 best: 0.4608426 (102)   total:
1.62s    remaining: 17.2s
103:    learn: 0.4433768        test: 0.4604916 best: 0.4604916 (103)   total:
1.63s    remaining: 17.2s
104:    learn: 0.4428260        test: 0.4601440 best: 0.4601440 (104)   total:
1.65s    remaining: 17.2s
105:    learn: 0.4419725        test: 0.4595507 best: 0.4595507 (105)   total:
1.67s    remaining: 17.2s
106:    learn: 0.4414475        test: 0.4592260 best: 0.4592260 (106)   total:
1.69s    remaining: 17.3s
107:    learn: 0.4409614        test: 0.4589600 best: 0.4589600 (107)   total:
1.72s    remaining: 17.4s
108:    learn: 0.4404546        test: 0.4586495 best: 0.4586495 (108)   total:
1.74s    remaining: 17.4s
109:    learn: 0.4400977        test: 0.4584212 best: 0.4584212 (109)   total:
1.76s    remaining: 17.4s
110:    learn: 0.4397297        test: 0.4582542 best: 0.4582542 (110)   total:
1.78s    remaining: 17.4s
111:    learn: 0.4392264        test: 0.4579408 best: 0.4579408 (111)   total:
1.79s    remaining: 17.4s
112:    learn: 0.4387337        test: 0.4576361 best: 0.4576361 (112)   total:
1.81s    remaining: 17.4s
113:    learn: 0.4379553        test: 0.4571004 best: 0.4571004 (113)   total:
1.82s    remaining: 17.4s
114:    learn: 0.4374574        test: 0.4568131 best: 0.4568131 (114)   total:
1.83s    remaining: 17.3s
115:    learn: 0.4367603        test: 0.4560688 best: 0.4560688 (115)   total:
1.85s    remaining: 17.3s
116:    learn: 0.4362112        test: 0.4556295 best: 0.4556295 (116)   total:
1.87s    remaining: 17.3s
117:    learn: 0.4360255        test: 0.4554157 best: 0.4554157 (117)   total:
1.88s    remaining: 17.2s
118:    learn: 0.4355748        test: 0.4550699 best: 0.4550699 (118)   total:
1.9s     remaining: 17.2s
119:    learn: 0.4350039        test: 0.4544661 best: 0.4544661 (119)   total:
1.92s    remaining: 17.2s
120:    learn: 0.4342915        test: 0.4539819 best: 0.4539819 (120)   total:
1.93s    remaining: 17.2s
121:    learn: 0.4336674        test: 0.4532966 best: 0.4532966 (121)   total:
1.96s    remaining: 17.3s
122:    learn: 0.4331223        test: 0.4527089 best: 0.4527089 (122)   total:
```

```
1.97s    remaining: 17.3s
123:    learn: 0.4327579        test: 0.4525299 best: 0.4525299 (123)   total:
1.99s    remaining: 17.3s
124:    learn: 0.4324061        test: 0.4522282 best: 0.4522282 (124)   total:
2s       remaining: 17.3s
125:    learn: 0.4317439        test: 0.4517847 best: 0.4517847 (125)   total:
2.02s    remaining: 17.2s
126:    learn: 0.4310979        test: 0.4513545 best: 0.4513545 (126)   total:
2.04s    remaining: 17.3s
127:    learn: 0.4307233        test: 0.4511273 best: 0.4511273 (127)   total:
2.06s    remaining: 17.3s
128:    learn: 0.4304110        test: 0.4506524 best: 0.4506524 (128)   total:
2.07s    remaining: 17.2s
129:    learn: 0.4298580        test: 0.4500385 best: 0.4500385 (129)   total:
2.08s    remaining: 17.2s
130:    learn: 0.4296560        test: 0.4499699 best: 0.4499699 (130)   total:
2.1s     remaining: 17.1s
131:    learn: 0.4294218        test: 0.4499111 best: 0.4499111 (131)   total:
2.11s    remaining: 17s
132:    learn: 0.4290545        test: 0.4495628 best: 0.4495628 (132)   total:
2.12s    remaining: 17s
133:    learn: 0.4285068        test: 0.4491996 best: 0.4491996 (133)   total:
2.13s    remaining: 16.9s
134:    learn: 0.4281694        test: 0.4488262 best: 0.4488262 (134)   total:
2.14s    remaining: 16.9s
135:    learn: 0.4275928        test: 0.4484508 best: 0.4484508 (135)   total:
2.15s    remaining: 16.8s
136:    learn: 0.4272273        test: 0.4481603 best: 0.4481603 (136)   total:
2.16s    remaining: 16.8s
137:    learn: 0.4269504        test: 0.4479124 best: 0.4479124 (137)   total:
2.17s    remaining: 16.7s
138:    learn: 0.4265304        test: 0.4476646 best: 0.4476646 (138)   total:
2.19s    remaining: 16.7s
139:    learn: 0.4261015        test: 0.4472472 best: 0.4472472 (139)   total:
2.2s     remaining: 16.7s
140:    learn: 0.4255652        test: 0.4469034 best: 0.4469034 (140)   total:
2.21s    remaining: 16.6s
141:    learn: 0.4252601        test: 0.4467439 best: 0.4467439 (141)   total:
2.22s    remaining: 16.6s
142:    learn: 0.4249331        test: 0.4465580 best: 0.4465580 (142)   total:
2.23s    remaining: 16.5s
143:    learn: 0.4246809        test: 0.4464741 best: 0.4464741 (143)   total:
2.25s    remaining: 16.5s
144:    learn: 0.4243615        test: 0.4462936 best: 0.4462936 (144)   total:
2.26s    remaining: 16.4s
145:    learn: 0.4241056        test: 0.4461492 best: 0.4461492 (145)   total:
2.27s    remaining: 16.4s
146:    learn: 0.4238803        test: 0.4459145 best: 0.4459145 (146)   total:
```

```
2.29s    remaining: 16.4s
147:    learn: 0.4236076        test: 0.4456295 best: 0.4456295 (147)   total:
2.31s    remaining: 16.4s
148:    learn: 0.4234581        test: 0.4456134 best: 0.4456134 (148)   total:
2.33s    remaining: 16.4s
149:    learn: 0.4232550        test: 0.4455049 best: 0.4455049 (149)   total:
2.34s    remaining: 16.4s
150:    learn: 0.4230303        test: 0.4452849 best: 0.4452849 (150)   total:
2.35s    remaining: 16.3s
151:    learn: 0.4226224        test: 0.4450084 best: 0.4450084 (151)   total:
2.36s    remaining: 16.3s
152:    learn: 0.4221786        test: 0.4445185 best: 0.4445185 (152)   total:
2.38s    remaining: 16.3s
153:    learn: 0.4218545        test: 0.4442526 best: 0.4442526 (153)   total:
2.39s    remaining: 16.2s
154:    learn: 0.4214268        test: 0.4439849 best: 0.4439849 (154)   total:
2.4s     remaining: 16.2s
155:    learn: 0.4210489        test: 0.4437612 best: 0.4437612 (155)   total:
2.41s    remaining: 16.2s
156:    learn: 0.4207866        test: 0.4436795 best: 0.4436795 (156)   total:
2.42s    remaining: 16.1s
157:    learn: 0.4205253        test: 0.4433940 best: 0.4433940 (157)   total:
2.44s    remaining: 16.1s
158:    learn: 0.4203173        test: 0.4432913 best: 0.4432913 (158)   total:
2.45s    remaining: 16.1s
159:    learn: 0.4202079        test: 0.4431933 best: 0.4431933 (159)   total:
2.47s    remaining: 16.1s
160:    learn: 0.4200230        test: 0.4431153 best: 0.4431153 (160)   total:
2.49s    remaining: 16.1s
161:    learn: 0.4196762        test: 0.4429600 best: 0.4429600 (161)   total:
2.51s    remaining: 16.1s
162:    learn: 0.4192563        test: 0.4427020 best: 0.4427020 (162)   total:
2.52s    remaining: 16s
163:    learn: 0.4189352        test: 0.4424271 best: 0.4424271 (163)   total:
2.54s    remaining: 16s
164:    learn: 0.4188244        test: 0.4424671 best: 0.4424271 (163)   total:
2.55s    remaining: 16s
165:    learn: 0.4183830        test: 0.4420117 best: 0.4420117 (165)   total:
2.56s    remaining: 16s
166:    learn: 0.4182698        test: 0.4420022 best: 0.4420022 (166)   total:
2.58s    remaining: 15.9s
167:    learn: 0.4178828        test: 0.4416209 best: 0.4416209 (167)   total:
2.59s    remaining: 15.9s
168:    learn: 0.4176269        test: 0.4414964 best: 0.4414964 (168)   total:
2.6s     remaining: 15.9s
169:    learn: 0.4173627        test: 0.4413661 best: 0.4413661 (169)   total:
2.62s    remaining: 15.9s
170:    learn: 0.4171304        test: 0.4411526 best: 0.4411526 (170)   total:
```

```
2.63s    remaining: 15.8s
171:    learn: 0.4169019        test: 0.4410948 best: 0.4410948 (171)   total:
2.64s    remaining: 15.8s
172:    learn: 0.4161666        test: 0.4393540 best: 0.4393540 (172)   total:
2.65s    remaining: 15.7s
173:    learn: 0.4154669        test: 0.4377259 best: 0.4377259 (173)   total:
2.66s    remaining: 15.7s
174:    learn: 0.4153516        test: 0.4377205 best: 0.4377205 (174)   total:
2.67s    remaining: 15.7s
175:    learn: 0.4150626        test: 0.4375076 best: 0.4375076 (175)   total:
2.69s    remaining: 15.6s
176:    learn: 0.4148373        test: 0.4373352 best: 0.4373352 (176)   total:
2.7s     remaining: 15.6s
177:    learn: 0.4144381        test: 0.4369147 best: 0.4369147 (177)   total:
2.71s    remaining: 15.6s
178:    learn: 0.4143469        test: 0.4369191 best: 0.4369147 (177)   total:
2.72s    remaining: 15.5s
179:    learn: 0.4141056        test: 0.4366601 best: 0.4366601 (179)   total:
2.73s    remaining: 15.5s
180:    learn: 0.4139240        test: 0.4363763 best: 0.4363763 (180)   total:
2.74s    remaining: 15.5s
181:    learn: 0.4136948        test: 0.4362535 best: 0.4362535 (181)   total:
2.76s    remaining: 15.4s
182:    learn: 0.4136263        test: 0.4362790 best: 0.4362535 (181)   total:
2.78s    remaining: 15.4s
183:    learn: 0.4133754        test: 0.4360726 best: 0.4360726 (183)   total:
2.79s    remaining: 15.4s
184:    learn: 0.4131700        test: 0.4359624 best: 0.4359624 (184)   total:
2.81s    remaining: 15.4s
185:    learn: 0.4129463        test: 0.4358645 best: 0.4358645 (185)   total:
2.82s    remaining: 15.4s
186:    learn: 0.4128337        test: 0.4359220 best: 0.4358645 (185)   total:
2.83s    remaining: 15.3s
187:    learn: 0.4126659        test: 0.4357082 best: 0.4357082 (187)   total:
2.84s    remaining: 15.3s
188:    learn: 0.4124963        test: 0.4354371 best: 0.4354371 (188)   total:
2.85s    remaining: 15.3s
189:    learn: 0.4121917        test: 0.4350654 best: 0.4350654 (189)   total:
2.87s    remaining: 15.3s
190:    learn: 0.4119051        test: 0.4349313 best: 0.4349313 (190)   total:
2.9s     remaining: 15.3s
191:    learn: 0.4115791        test: 0.4347395 best: 0.4347395 (191)   total:
2.91s    remaining: 15.3s
192:    learn: 0.4113900        test: 0.4346547 best: 0.4346547 (192)   total:
2.93s    remaining: 15.3s
193:    learn: 0.4112483        test: 0.4345802 best: 0.4345802 (193)   total:
2.94s    remaining: 15.2s
194:    learn: 0.4109036        test: 0.4342134 best: 0.4342134 (194)   total:
```

```
2.95s    remaining: 15.2s
195:    learn: 0.4107163        test: 0.4341800 best: 0.4341800 (195)   total:
2.97s    remaining: 15.2s
196:    learn: 0.4105595        test: 0.4339254 best: 0.4339254 (196)   total:
2.98s    remaining: 15.2s
197:    learn: 0.4102515        test: 0.4337458 best: 0.4337458 (197)   total:
3s       remaining: 15.2s
198:    learn: 0.4100527        test: 0.4336958 best: 0.4336958 (198)   total:
3.01s    remaining: 15.1s
199:    learn: 0.4098683        test: 0.4333796 best: 0.4333796 (199)   total:
3.02s    remaining: 15.1s
200:    learn: 0.4097365        test: 0.4332640 best: 0.4332640 (200)   total:
3.04s    remaining: 15.1s
201:    learn: 0.4095475        test: 0.4330037 best: 0.4330037 (201)   total:
3.06s    remaining: 15.1s
202:    learn: 0.4094533        test: 0.4329556 best: 0.4329556 (202)   total:
3.08s    remaining: 15.1s
203:    learn: 0.4092400        test: 0.4327717 best: 0.4327717 (203)   total:
3.1s     remaining: 15.1s
204:    learn: 0.4089182        test: 0.4324266 best: 0.4324266 (204)   total:
3.11s    remaining: 15.1s
205:    learn: 0.4088437        test: 0.4324018 best: 0.4324018 (205)   total:
3.12s    remaining: 15.1s
206:    learn: 0.4086705        test: 0.4323358 best: 0.4323358 (206)   total:
3.13s    remaining: 15s
207:    learn: 0.4085052        test: 0.4323421 best: 0.4323358 (206)   total:
3.15s    remaining: 15s
208:    learn: 0.4083645        test: 0.4322645 best: 0.4322645 (208)   total:
3.17s    remaining: 15s
209:    learn: 0.4082598        test: 0.4322689 best: 0.4322645 (208)   total:
3.18s    remaining: 15s
210:    learn: 0.4081316        test: 0.4322459 best: 0.4322459 (210)   total:
3.2s     remaining: 15s
211:    learn: 0.4081243        test: 0.4322817 best: 0.4322459 (210)   total:
3.21s    remaining: 15s
212:    learn: 0.4080651        test: 0.4322742 best: 0.4322459 (210)   total:
3.23s    remaining: 15s
213:    learn: 0.4078801        test: 0.4321897 best: 0.4321897 (213)   total:
3.24s    remaining: 14.9s
214:    learn: 0.4076542        test: 0.4320047 best: 0.4320047 (214)   total:
3.25s    remaining: 14.9s
215:    learn: 0.4074294        test: 0.4317500 best: 0.4317500 (215)   total:
3.27s    remaining: 14.9s
216:    learn: 0.4073026        test: 0.4316794 best: 0.4316794 (216)   total:
3.28s    remaining: 14.9s
217:    learn: 0.4070824        test: 0.4315202 best: 0.4315202 (217)   total:
3.29s    remaining: 14.8s
218:    learn: 0.4069618        test: 0.4315140 best: 0.4315140 (218)   total:
```

```
3.31s    remaining: 14.8s
219:    learn: 0.4068474        test: 0.4314484 best: 0.4314484 (219)   total:
3.32s    remaining: 14.8s
220:    learn: 0.4067314        test: 0.4313445 best: 0.4313445 (220)   total:
3.33s    remaining: 14.8s
221:    learn: 0.4065055        test: 0.4312077 best: 0.4312077 (221)   total:
3.34s    remaining: 14.7s
222:    learn: 0.4063142        test: 0.4310381 best: 0.4310381 (222)   total:
3.36s    remaining: 14.7s
223:    learn: 0.4061330        test: 0.4309832 best: 0.4309832 (223)   total:
3.37s    remaining: 14.7s
224:    learn: 0.4060000        test: 0.4308849 best: 0.4308849 (224)   total:
3.38s    remaining: 14.7s
225:    learn: 0.4057762        test: 0.4307112 best: 0.4307112 (225)   total:
3.4s     remaining: 14.6s
226:    learn: 0.4056301        test: 0.4307119 best: 0.4307112 (225)   total:
3.41s    remaining: 14.6s
227:    learn: 0.4054968        test: 0.4306740 best: 0.4306740 (227)   total:
3.42s    remaining: 14.6s
228:    learn: 0.4054304        test: 0.4306049 best: 0.4306049 (228)   total:
3.44s    remaining: 14.6s
229:    learn: 0.4051330        test: 0.4303063 best: 0.4303063 (229)   total:
3.45s    remaining: 14.6s
230:    learn: 0.4050330        test: 0.4302498 best: 0.4302498 (230)   total:
3.46s    remaining: 14.5s
231:    learn: 0.4047456        test: 0.4301056 best: 0.4301056 (231)   total:
3.48s    remaining: 14.5s
232:    learn: 0.4046363        test: 0.4300538 best: 0.4300538 (232)   total:
3.5s     remaining: 14.5s
233:    learn: 0.4045200        test: 0.4300466 best: 0.4300466 (233)   total:
3.51s    remaining: 14.5s
234:    learn: 0.4044211        test: 0.4299902 best: 0.4299902 (234)   total:
3.52s    remaining: 14.5s
235:    learn: 0.4042892        test: 0.4299430 best: 0.4299430 (235)   total:
3.53s    remaining: 14.4s
236:    learn: 0.4041467        test: 0.4298910 best: 0.4298910 (236)   total:
3.55s    remaining: 14.4s
237:    learn: 0.4040835        test: 0.4298607 best: 0.4298607 (237)   total:
3.56s    remaining: 14.4s
238:    learn: 0.4038428        test: 0.4296090 best: 0.4296090 (238)   total:
3.58s    remaining: 14.4s
239:    learn: 0.4036164        test: 0.4293344 best: 0.4293344 (239)   total:
3.59s    remaining: 14.4s
240:    learn: 0.4034539        test: 0.4291535 best: 0.4291535 (240)   total:
3.6s     remaining: 14.3s
241:    learn: 0.4033064        test: 0.4289298 best: 0.4289298 (241)   total:
3.61s    remaining: 14.3s
242:    learn: 0.4031781        test: 0.4288931 best: 0.4288931 (242)   total:
```

```
3.62s    remaining: 14.3s
243:    learn: 0.4029580        test: 0.4288328 best: 0.4288328 (243)   total:
3.63s    remaining: 14.2s
244:    learn: 0.4028801        test: 0.4287704 best: 0.4287704 (244)   total:
3.65s    remaining: 14.2s
245:    learn: 0.4026705        test: 0.4286529 best: 0.4286529 (245)   total:
3.66s    remaining: 14.2s
246:    learn: 0.4025324        test: 0.4284852 best: 0.4284852 (246)   total:
3.67s    remaining: 14.2s
247:    learn: 0.4020323        test: 0.4272021 best: 0.4272021 (247)   total:
3.69s    remaining: 14.2s
248:    learn: 0.4018933        test: 0.4270018 best: 0.4270018 (248)   total:
3.7s     remaining: 14.1s
249:    learn: 0.4018013        test: 0.4270328 best: 0.4270018 (248)   total:
3.71s    remaining: 14.1s
250:    learn: 0.4017437        test: 0.4269795 best: 0.4269795 (250)   total:
3.73s    remaining: 14.1s
251:    learn: 0.4017113        test: 0.4269704 best: 0.4269704 (251)   total:
3.74s    remaining: 14.1s
252:    learn: 0.4016329        test: 0.4269108 best: 0.4269108 (252)   total:
3.75s    remaining: 14s
253:    learn: 0.4015834        test: 0.4269180 best: 0.4269108 (252)   total:
3.77s    remaining: 14s
254:    learn: 0.4014739        test: 0.4268694 best: 0.4268694 (254)   total:
3.78s    remaining: 14s
255:    learn: 0.4013723        test: 0.4268583 best: 0.4268583 (255)   total:
3.79s    remaining: 14s
256:    learn: 0.4012855        test: 0.4268059 best: 0.4268059 (256)   total:
3.81s    remaining: 14s
257:    learn: 0.4012384        test: 0.4267464 best: 0.4267464 (257)   total:
3.82s    remaining: 14s
258:    learn: 0.4010464        test: 0.4265719 best: 0.4265719 (258)   total:
3.84s    remaining: 13.9s
259:    learn: 0.4008061        test: 0.4263026 best: 0.4263026 (259)   total:
3.85s    remaining: 13.9s
260:    learn: 0.4006038        test: 0.4261600 best: 0.4261600 (260)   total:
3.86s    remaining: 13.9s
261:    learn: 0.4005208        test: 0.4261099 best: 0.4261099 (261)   total:
3.88s    remaining: 13.9s
262:    learn: 0.4004751        test: 0.4261114 best: 0.4261099 (261)   total:
3.89s    remaining: 13.9s
263:    learn: 0.4002979        test: 0.4258643 best: 0.4258643 (263)   total:
3.9s     remaining: 13.8s
264:    learn: 0.4002645        test: 0.4258642 best: 0.4258642 (264)   total:
3.92s    remaining: 13.8s
265:    learn: 0.4000804        test: 0.4256961 best: 0.4256961 (265)   total:
3.93s    remaining: 13.8s
266:    learn: 0.3999086        test: 0.4254811 best: 0.4254811 (266)   total:
```

```
3.95s    remaining: 13.8s
267:    learn: 0.3998292        test: 0.4254328 best: 0.4254328 (267)   total:
3.96s    remaining: 13.8s
268:    learn: 0.3996502        test: 0.4252694 best: 0.4252694 (268)   total:
3.98s    remaining: 13.8s
269:    learn: 0.3994257        test: 0.4250178 best: 0.4250178 (269)   total:
3.99s    remaining: 13.7s
270:    learn: 0.3992437        test: 0.4249194 best: 0.4249194 (270)   total:
4.01s    remaining: 13.7s
271:    learn: 0.3992012        test: 0.4249156 best: 0.4249156 (271)   total:
4.02s    remaining: 13.7s
272:    learn: 0.3991056        test: 0.4248737 best: 0.4248737 (272)   total:
4.04s    remaining: 13.7s
273:    learn: 0.3990507        test: 0.4248474 best: 0.4248474 (273)   total:
4.05s    remaining: 13.7s
274:    learn: 0.3989596        test: 0.4248446 best: 0.4248446 (274)   total:
4.07s    remaining: 13.7s
275:    learn: 0.3989130        test: 0.4247917 best: 0.4247917 (275)   total:
4.08s    remaining: 13.7s
276:    learn: 0.3988203        test: 0.4247639 best: 0.4247639 (276)   total:
4.09s    remaining: 13.6s
277:    learn: 0.3983800        test: 0.4235839 best: 0.4235839 (277)   total:
4.1s     remaining: 13.6s
278:    learn: 0.3982229        test: 0.4234802 best: 0.4234802 (278)   total:
4.11s    remaining: 13.6s
279:    learn: 0.3981827        test: 0.4235368 best: 0.4234802 (278)   total:
4.13s    remaining: 13.6s
280:    learn: 0.3979334        test: 0.4232847 best: 0.4232847 (280)   total:
4.14s    remaining: 13.5s
281:    learn: 0.3977765        test: 0.4231989 best: 0.4231989 (281)   total:
4.15s    remaining: 13.5s
282:    learn: 0.3977329        test: 0.4231499 best: 0.4231499 (282)   total:
4.17s    remaining: 13.5s
283:    learn: 0.3976876        test: 0.4230805 best: 0.4230805 (283)   total:
4.18s    remaining: 13.5s
284:    learn: 0.3974904        test: 0.4229253 best: 0.4229253 (284)   total:
4.2s     remaining: 13.5s
285:    learn: 0.3972565        test: 0.4228279 best: 0.4228279 (285)   total:
4.21s    remaining: 13.5s
286:    learn: 0.3972205        test: 0.4228448 best: 0.4228279 (285)   total:
4.22s    remaining: 13.4s
287:    learn: 0.3970184        test: 0.4226130 best: 0.4226130 (287)   total:
4.24s    remaining: 13.4s
288:    learn: 0.3968984        test: 0.4225669 best: 0.4225669 (288)   total:
4.25s    remaining: 13.4s
289:    learn: 0.3967677        test: 0.4225250 best: 0.4225250 (289)   total:
4.26s    remaining: 13.4s
290:    learn: 0.3966231        test: 0.4225357 best: 0.4225250 (289)   total:
```

```
4.28s    remaining: 13.4s
291:    learn: 0.3966207        test: 0.4225574 best: 0.4225250 (289)   total:
4.3s     remaining: 13.4s
292:    learn: 0.3965481        test: 0.4225238 best: 0.4225238 (292)   total:
4.31s    remaining: 13.4s
293:    learn: 0.3965269        test: 0.4225493 best: 0.4225238 (292)   total:
4.33s    remaining: 13.3s
294:    learn: 0.3964947        test: 0.4225590 best: 0.4225238 (292)   total:
4.34s    remaining: 13.3s
295:    learn: 0.3964159        test: 0.4225462 best: 0.4225238 (292)   total:
4.35s    remaining: 13.3s
296:    learn: 0.3963563        test: 0.4225574 best: 0.4225238 (292)   total:
4.36s    remaining: 13.3s
297:    learn: 0.3962873        test: 0.4225461 best: 0.4225238 (292)   total:
4.37s    remaining: 13.2s
298:    learn: 0.3962369        test: 0.4224872 best: 0.4224872 (298)   total:
4.38s    remaining: 13.2s
299:    learn: 0.3961690        test: 0.4224421 best: 0.4224421 (299)   total:
4.4s     remaining: 13.2s
300:    learn: 0.3960340        test: 0.4223629 best: 0.4223629 (300)   total:
4.41s    remaining: 13.2s
301:    learn: 0.3959792        test: 0.4222858 best: 0.4222858 (301)   total:
4.43s    remaining: 13.2s
302:    learn: 0.3958925        test: 0.4222188 best: 0.4222188 (302)   total:
4.44s    remaining: 13.1s
303:    learn: 0.3956566        test: 0.4219812 best: 0.4219812 (303)   total:
4.45s    remaining: 13.1s
304:    learn: 0.3954903        test: 0.4222050 best: 0.4219812 (303)   total:
4.46s    remaining: 13.1s
305:    learn: 0.3954596        test: 0.4222212 best: 0.4219812 (303)   total:
4.48s    remaining: 13.1s
306:    learn: 0.3954103        test: 0.4222077 best: 0.4219812 (303)   total:
4.49s    remaining: 13.1s
307:    learn: 0.3952196        test: 0.4220018 best: 0.4219812 (303)   total:
4.51s    remaining: 13.1s
308:    learn: 0.3949602        test: 0.4219569 best: 0.4219569 (308)   total:
4.52s    remaining: 13s
309:    learn: 0.3948982        test: 0.4219421 best: 0.4219421 (309)   total:
4.54s    remaining: 13s
310:    learn: 0.3948699        test: 0.4219221 best: 0.4219221 (310)   total:
4.55s    remaining: 13s
311:    learn: 0.3947261        test: 0.4217228 best: 0.4217228 (311)   total:
4.56s    remaining: 13s
312:    learn: 0.3947040        test: 0.4217349 best: 0.4217228 (311)   total:
4.58s    remaining: 13s
313:    learn: 0.3945209        test: 0.4215950 best: 0.4215950 (313)   total:
4.6s     remaining: 13s
314:    learn: 0.3944962        test: 0.4216118 best: 0.4215950 (313)   total:
```

```
4.61s    remaining: 13s
315:    learn: 0.3943540        test: 0.4214179 best: 0.4214179 (315)   total:
4.63s    remaining: 12.9s
316:    learn: 0.3942437        test: 0.4213174 best: 0.4213174 (316)   total:
4.65s    remaining: 12.9s
317:    learn: 0.3941059        test: 0.4211252 best: 0.4211252 (317)   total:
4.67s    remaining: 12.9s
318:    learn: 0.3940297        test: 0.4210693 best: 0.4210693 (318)   total:
4.69s    remaining: 12.9s
319:    learn: 0.3938096        test: 0.4208487 best: 0.4208487 (319)   total:
4.7s     remaining: 12.9s
320:    learn: 0.3937383        test: 0.4207427 best: 0.4207427 (320)   total:
4.72s    remaining: 12.9s
321:    learn: 0.3936735        test: 0.4207340 best: 0.4207340 (321)   total:
4.73s    remaining: 12.9s
322:    learn: 0.3936124        test: 0.4206749 best: 0.4206749 (322)   total:
4.74s    remaining: 12.9s
323:    learn: 0.3935809        test: 0.4206667 best: 0.4206667 (323)   total:
4.75s    remaining: 12.9s
324:    learn: 0.3935464        test: 0.4206821 best: 0.4206667 (323)   total:
4.77s    remaining: 12.8s
325:    learn: 0.3935277        test: 0.4206845 best: 0.4206667 (323)   total:
4.78s    remaining: 12.8s
326:    learn: 0.3934659        test: 0.4206680 best: 0.4206667 (323)   total:
4.8s     remaining: 12.8s
327:    learn: 0.3933320        test: 0.4204835 best: 0.4204835 (327)   total:
4.81s    remaining: 12.8s
328:    learn: 0.3932591        test: 0.4204760 best: 0.4204760 (328)   total:
4.82s    remaining: 12.8s
329:    learn: 0.3931578        test: 0.4204378 best: 0.4204378 (329)   total:
4.83s    remaining: 12.7s
330:    learn: 0.3930923        test: 0.4204052 best: 0.4204052 (330)   total:
4.84s    remaining: 12.7s
331:    learn: 0.3929745        test: 0.4203954 best: 0.4203954 (331)   total:
4.85s    remaining: 12.7s
332:    learn: 0.3928919        test: 0.4204081 best: 0.4203954 (331)   total:
4.87s    remaining: 12.7s
333:    learn: 0.3928204        test: 0.4203864 best: 0.4203864 (333)   total:
4.88s    remaining: 12.7s
334:    learn: 0.3927992        test: 0.4203908 best: 0.4203864 (333)   total:
4.89s    remaining: 12.6s
335:    learn: 0.3927484        test: 0.4203721 best: 0.4203721 (335)   total:
4.9s     remaining: 12.6s
336:    learn: 0.3926947        test: 0.4203325 best: 0.4203325 (336)   total:
4.91s    remaining: 12.6s
337:    learn: 0.3926072        test: 0.4201798 best: 0.4201798 (337)   total:
4.92s    remaining: 12.6s
338:    learn: 0.3925797        test: 0.4201695 best: 0.4201695 (338)   total:
```

```
4.94s    remaining: 12.6s
339:    learn: 0.3925132        test: 0.4201623 best: 0.4201623 (339)   total:
4.96s    remaining: 12.5s
340:    learn: 0.3924449        test: 0.4200163 best: 0.4200163 (340)   total:
4.97s    remaining: 12.5s
341:    learn: 0.3923100        test: 0.4198772 best: 0.4198772 (341)   total:
4.98s    remaining: 12.5s
342:    learn: 0.3922063        test: 0.4196492 best: 0.4196492 (342)   total:
5s       remaining: 12.5s
343:    learn: 0.3921658        test: 0.4196437 best: 0.4196437 (343)   total:
5.02s    remaining: 12.5s
344:    learn: 0.3920995        test: 0.4195945 best: 0.4195945 (344)   total:
5.03s    remaining: 12.5s
345:    learn: 0.3920752        test: 0.4195723 best: 0.4195723 (345)   total:
5.05s    remaining: 12.5s
346:    learn: 0.3920283        test: 0.4195008 best: 0.4195008 (346)   total:
5.06s    remaining: 12.4s
347:    learn: 0.3919653        test: 0.4194865 best: 0.4194865 (347)   total:
5.07s    remaining: 12.4s
348:    learn: 0.3919220        test: 0.4194910 best: 0.4194865 (347)   total:
5.08s    remaining: 12.4s
349:    learn: 0.3918707        test: 0.4194500 best: 0.4194500 (349)   total:
5.1s     remaining: 12.4s
350:    learn: 0.3918139        test: 0.4194021 best: 0.4194021 (350)   total:
5.11s    remaining: 12.4s
351:    learn: 0.3917250        test: 0.4193230 best: 0.4193230 (351)   total:
5.12s    remaining: 12.3s
352:    learn: 0.3916241        test: 0.4191808 best: 0.4191808 (352)   total:
5.13s    remaining: 12.3s
353:    learn: 0.3914577        test: 0.4189974 best: 0.4189974 (353)   total:
5.14s    remaining: 12.3s
354:    learn: 0.3913549        test: 0.4189524 best: 0.4189524 (354)   total:
5.16s    remaining: 12.3s
355:    learn: 0.3913437        test: 0.4189297 best: 0.4189297 (355)   total:
5.17s    remaining: 12.3s
356:    learn: 0.3912317        test: 0.4188709 best: 0.4188709 (356)   total:
5.18s    remaining: 12.2s
357:    learn: 0.3911809        test: 0.4188325 best: 0.4188325 (357)   total:
5.19s    remaining: 12.2s
358:    learn: 0.3910191        test: 0.4186530 best: 0.4186530 (358)   total:
5.2s     remaining: 12.2s
359:    learn: 0.3909795        test: 0.4186973 best: 0.4186530 (358)   total:
5.22s    remaining: 12.2s
360:    learn: 0.3909208        test: 0.4186431 best: 0.4186431 (360)   total:
5.23s    remaining: 12.2s
361:    learn: 0.3908399        test: 0.4185155 best: 0.4185155 (361)   total:
5.25s    remaining: 12.1s
362:    learn: 0.3907688        test: 0.4184678 best: 0.4184678 (362)   total:
```

```
5.26s    remaining: 12.1s
363:    learn: 0.3906598        test: 0.4184118 best: 0.4184118 (363)   total:
5.28s    remaining: 12.1s
364:    learn: 0.3906053        test: 0.4184113 best: 0.4184113 (364)   total:
5.29s    remaining: 12.1s
365:    learn: 0.3905490        test: 0.4184036 best: 0.4184036 (365)   total:
5.31s    remaining: 12.1s
366:    learn: 0.3905113        test: 0.4183789 best: 0.4183789 (366)   total:
5.32s    remaining: 12.1s
367:    learn: 0.3904839        test: 0.4183397 best: 0.4183397 (367)   total:
5.34s    remaining: 12.1s
368:    learn: 0.3903846        test: 0.4182307 best: 0.4182307 (368)   total:
5.35s    remaining: 12.1s
369:    learn: 0.3902913        test: 0.4181875 best: 0.4181875 (369)   total:
5.36s    remaining: 12s
370:    learn: 0.3902906        test: 0.4181932 best: 0.4181875 (369)   total:
5.38s    remaining: 12s
371:    learn: 0.3902666        test: 0.4181856 best: 0.4181856 (371)   total:
5.39s    remaining: 12s
372:    learn: 0.3902500        test: 0.4181910 best: 0.4181856 (371)   total:
5.4s     remaining: 12s
373:    learn: 0.3900231        test: 0.4181522 best: 0.4181522 (373)   total:
5.43s    remaining: 12s
374:    learn: 0.3898813        test: 0.4180436 best: 0.4180436 (374)   total:
5.45s    remaining: 12s
375:    learn: 0.3898302        test: 0.4180431 best: 0.4180431 (375)   total:
5.47s    remaining: 12s
376:    learn: 0.3898063        test: 0.4180473 best: 0.4180431 (375)   total:
5.48s    remaining: 12s
377:    learn: 0.3897824        test: 0.4180443 best: 0.4180431 (375)   total:
5.5s     remaining: 12s
378:    learn: 0.3897644        test: 0.4180570 best: 0.4180431 (375)   total:
5.52s    remaining: 11.9s
379:    learn: 0.3897389        test: 0.4180603 best: 0.4180431 (375)   total:
5.53s    remaining: 11.9s
380:    learn: 0.3897267        test: 0.4180630 best: 0.4180431 (375)   total:
5.54s    remaining: 11.9s
381:    learn: 0.3897035        test: 0.4180562 best: 0.4180431 (375)   total:
5.55s    remaining: 11.9s
382:    learn: 0.3896092        test: 0.4179685 best: 0.4179685 (382)   total:
5.57s    remaining: 11.9s
383:    learn: 0.3895558        test: 0.4179279 best: 0.4179279 (383)   total:
5.58s    remaining: 11.9s
384:    learn: 0.3895337        test: 0.4179362 best: 0.4179279 (383)   total:
5.59s    remaining: 11.8s
385:    learn: 0.3894070        test: 0.4179332 best: 0.4179279 (383)   total:
5.61s    remaining: 11.8s
386:    learn: 0.3892886        test: 0.4178350 best: 0.4178350 (386)   total:
```

```
5.62s    remaining: 11.8s
387:    learn: 0.3889085        test: 0.4167029 best: 0.4167029 (387)   total:
5.63s    remaining: 11.8s
388:    learn: 0.3888910        test: 0.4166856 best: 0.4166856 (388)   total:
5.64s    remaining: 11.8s
389:    learn: 0.3888781        test: 0.4166690 best: 0.4166690 (389)   total:
5.66s    remaining: 11.7s
390:    learn: 0.3888515        test: 0.4166726 best: 0.4166690 (389)   total:
5.67s    remaining: 11.7s
391:    learn: 0.3888067        test: 0.4166365 best: 0.4166365 (391)   total:
5.68s    remaining: 11.7s
392:    learn: 0.3887628        test: 0.4166012 best: 0.4166012 (392)   total:
5.69s    remaining: 11.7s
393:    learn: 0.3887395        test: 0.4166055 best: 0.4166012 (392)   total:
5.7s     remaining: 11.7s
394:    learn: 0.3886607        test: 0.4164674 best: 0.4164674 (394)   total:
5.71s    remaining: 11.6s
395:    learn: 0.3886517        test: 0.4164755 best: 0.4164674 (394)   total:
5.72s    remaining: 11.6s
396:    learn: 0.3886374        test: 0.4164556 best: 0.4164556 (396)   total:
5.74s    remaining: 11.6s
397:    learn: 0.3885938        test: 0.4164203 best: 0.4164203 (397)   total:
5.75s    remaining: 11.6s
398:    learn: 0.3885202        test: 0.4162826 best: 0.4162826 (398)   total:
5.77s    remaining: 11.6s
399:    learn: 0.3884645        test: 0.4162503 best: 0.4162503 (399)   total:
5.78s    remaining: 11.6s
400:    learn: 0.3884106        test: 0.4161625 best: 0.4161625 (400)   total:
5.8s     remaining: 11.6s
401:    learn: 0.3882964        test: 0.4161662 best: 0.4161625 (400)   total:
5.81s    remaining: 11.5s
402:    learn: 0.3881339        test: 0.4161812 best: 0.4161625 (400)   total:
5.82s    remaining: 11.5s
403:    learn: 0.3879432        test: 0.4159902 best: 0.4159902 (403)   total:
5.83s    remaining: 11.5s
404:    learn: 0.3877517        test: 0.4158974 best: 0.4158974 (404)   total:
5.84s    remaining: 11.5s
405:    learn: 0.3876603        test: 0.4157466 best: 0.4157466 (405)   total:
5.85s    remaining: 11.4s
406:    learn: 0.3875708        test: 0.4157026 best: 0.4157026 (406)   total:
5.88s    remaining: 11.4s
407:    learn: 0.3875338        test: 0.4156753 best: 0.4156753 (407)   total:
5.89s    remaining: 11.4s
408:    learn: 0.3875115        test: 0.4156684 best: 0.4156684 (408)   total:
5.9s     remaining: 11.4s
409:    learn: 0.3874052        test: 0.4155066 best: 0.4155066 (409)   total:
5.92s    remaining: 11.4s
410:    learn: 0.3873565        test: 0.4155045 best: 0.4155045 (410)   total:
```

```
5.93s    remaining: 11.4s
411:    learn: 0.3871733        test: 0.4153221 best: 0.4153221 (411)   total:
5.94s    remaining: 11.4s
412:    learn: 0.3871124        test: 0.4153192 best: 0.4153192 (412)   total:
5.98s    remaining: 11.4s
413:    learn: 0.3870729        test: 0.4152873 best: 0.4152873 (413)   total:
6s       remaining: 11.4s
414:    learn: 0.3870476        test: 0.4152818 best: 0.4152818 (414)   total:
6.01s    remaining: 11.4s
415:    learn: 0.3870235        test: 0.4152728 best: 0.4152728 (415)   total:
6.02s    remaining: 11.3s
416:    learn: 0.3869817        test: 0.4152598 best: 0.4152598 (416)   total:
6.04s    remaining: 11.3s
417:    learn: 0.3866202        test: 0.4142120 best: 0.4142120 (417)   total:
6.06s    remaining: 11.3s
418:    learn: 0.3865802        test: 0.4141786 best: 0.4141786 (418)   total:
6.07s    remaining: 11.3s
419:    learn: 0.3865543        test: 0.4142048 best: 0.4141786 (418)   total:
6.08s    remaining: 11.3s
420:    learn: 0.3865240        test: 0.4142005 best: 0.4141786 (418)   total:
6.09s    remaining: 11.3s
421:    learn: 0.3864775        test: 0.4141343 best: 0.4141343 (421)   total:
6.11s    remaining: 11.3s
422:    learn: 0.3864561        test: 0.4141418 best: 0.4141343 (421)   total:
6.12s    remaining: 11.2s
423:    learn: 0.3864370        test: 0.4141490 best: 0.4141343 (421)   total:
6.14s    remaining: 11.2s
424:    learn: 0.3863215        test: 0.4140305 best: 0.4140305 (424)   total:
6.16s    remaining: 11.2s
425:    learn: 0.3862836        test: 0.4139999 best: 0.4139999 (425)   total:
6.18s    remaining: 11.2s
426:    learn: 0.3862490        test: 0.4139937 best: 0.4139937 (426)   total:
6.19s    remaining: 11.2s
427:    learn: 0.3862399        test: 0.4139716 best: 0.4139716 (427)   total:
6.21s    remaining: 11.2s
428:    learn: 0.3862348        test: 0.4139849 best: 0.4139716 (427)   total:
6.22s    remaining: 11.2s
429:    learn: 0.3861996        test: 0.4139459 best: 0.4139459 (429)   total:
6.23s    remaining: 11.2s
430:    learn: 0.3861615        test: 0.4139104 best: 0.4139104 (430)   total:
6.25s    remaining: 11.1s
431:    learn: 0.3861391        test: 0.4139152 best: 0.4139104 (430)   total:
6.26s    remaining: 11.1s
432:    learn: 0.3861018        test: 0.4138837 best: 0.4138837 (432)   total:
6.28s    remaining: 11.1s
433:    learn: 0.3860907        test: 0.4138954 best: 0.4138837 (432)   total:
6.29s    remaining: 11.1s
434:    learn: 0.3860333        test: 0.4137917 best: 0.4137917 (434)   total:
```

```
6.31s    remaining: 11.1s
435:    learn: 0.3859965        test: 0.4137604 best: 0.4137604 (435)   total:
6.33s    remaining: 11.1s
436:    learn: 0.3859022        test: 0.4136550 best: 0.4136550 (436)   total:
6.35s    remaining: 11.1s
437:    learn: 0.3858403        test: 0.4136194 best: 0.4136194 (437)   total:
6.36s    remaining: 11.1s
438:    learn: 0.3858368        test: 0.4136334 best: 0.4136194 (437)   total:
6.37s    remaining: 11s
439:    learn: 0.3858013        test: 0.4136043 best: 0.4136043 (439)   total:
6.38s    remaining: 11s
440:    learn: 0.3855483        test: 0.4128211 best: 0.4128211 (440)   total:
6.4s     remaining: 11s
441:    learn: 0.3854816        test: 0.4127766 best: 0.4127766 (441)   total:
6.41s    remaining: 11s
442:    learn: 0.3854621        test: 0.4127844 best: 0.4127766 (441)   total:
6.42s    remaining: 11s
443:    learn: 0.3853407        test: 0.4127104 best: 0.4127104 (443)   total:
6.44s    remaining: 11s
444:    learn: 0.3852875        test: 0.4127147 best: 0.4127104 (443)   total:
6.45s    remaining: 10.9s
445:    learn: 0.3852844        test: 0.4127044 best: 0.4127044 (445)   total:
6.46s    remaining: 10.9s
446:    learn: 0.3851835        test: 0.4126363 best: 0.4126363 (446)   total:
6.48s    remaining: 10.9s
447:    learn: 0.3851031        test: 0.4126067 best: 0.4126067 (447)   total:
6.49s    remaining: 10.9s
448:    learn: 0.3850336        test: 0.4125554 best: 0.4125554 (448)   total:
6.5s     remaining: 10.9s
449:    learn: 0.3850041        test: 0.4125502 best: 0.4125502 (449)   total:
6.51s    remaining: 10.9s
450:    learn: 0.3848941        test: 0.4125676 best: 0.4125502 (449)   total:
6.53s    remaining: 10.8s
451:    learn: 0.3848092        test: 0.4125145 best: 0.4125145 (451)   total:
6.54s    remaining: 10.8s
452:    learn: 0.3847729        test: 0.4124880 best: 0.4124880 (452)   total:
6.55s    remaining: 10.8s
453:    learn: 0.3847479        test: 0.4124879 best: 0.4124879 (453)   total:
6.56s    remaining: 10.8s
454:    learn: 0.3847460        test: 0.4124933 best: 0.4124879 (453)   total:
6.57s    remaining: 10.8s
455:    learn: 0.3847325        test: 0.4124962 best: 0.4124879 (453)   total:
6.58s    remaining: 10.7s
456:    learn: 0.3846994        test: 0.4124685 best: 0.4124685 (456)   total:
6.6s     remaining: 10.7s
457:    learn: 0.3846926        test: 0.4124739 best: 0.4124685 (456)   total:
6.61s    remaining: 10.7s
458:    learn: 0.3846782        test: 0.4124671 best: 0.4124671 (458)   total:
```

```
6.63s    remaining: 10.7s
459:    learn: 0.3846686        test: 0.4124550 best: 0.4124550 (459)   total:
6.64s    remaining: 10.7s
460:    learn: 0.3846296        test: 0.4124185 best: 0.4124185 (460)   total:
6.66s    remaining: 10.7s
461:    learn: 0.3845830        test: 0.4123980 best: 0.4123980 (461)   total:
6.67s    remaining: 10.7s
462:    learn: 0.3844472        test: 0.4123308 best: 0.4123308 (462)   total:
6.68s    remaining: 10.6s
463:    learn: 0.3843373        test: 0.4122387 best: 0.4122387 (463)   total:
6.69s    remaining: 10.6s
464:    learn: 0.3842079        test: 0.4121799 best: 0.4121799 (464)   total:
6.7s     remaining: 10.6s
465:    learn: 0.3841765        test: 0.4121534 best: 0.4121534 (465)   total:
6.71s    remaining: 10.6s
466:    learn: 0.3841761        test: 0.4121562 best: 0.4121534 (465)   total:
6.73s    remaining: 10.6s
467:    learn: 0.3841188        test: 0.4121163 best: 0.4121163 (467)   total:
6.74s    remaining: 10.5s
468:    learn: 0.3841179        test: 0.4121177 best: 0.4121163 (467)   total:
6.75s    remaining: 10.5s
469:    learn: 0.3840764        test: 0.4121095 best: 0.4121095 (469)   total:
6.76s    remaining: 10.5s
470:    learn: 0.3840640        test: 0.4121152 best: 0.4121095 (469)   total:
6.78s    remaining: 10.5s
471:    learn: 0.3840383        test: 0.4120978 best: 0.4120978 (471)   total:
6.79s    remaining: 10.5s
472:    learn: 0.3840299        test: 0.4121088 best: 0.4120978 (471)   total:
6.81s    remaining: 10.5s
473:    learn: 0.3840013        test: 0.4120848 best: 0.4120848 (473)   total:
6.82s    remaining: 10.4s
474:    learn: 0.3839902        test: 0.4120966 best: 0.4120848 (473)   total:
6.83s    remaining: 10.4s
475:    learn: 0.3839193        test: 0.4121029 best: 0.4120848 (473)   total:
6.84s    remaining: 10.4s
476:    learn: 0.3839096        test: 0.4120779 best: 0.4120779 (476)   total:
6.85s    remaining: 10.4s
477:    learn: 0.3837704        test: 0.4120830 best: 0.4120779 (476)   total:
6.87s    remaining: 10.4s
478:    learn: 0.3837569        test: 0.4120918 best: 0.4120779 (476)   total:
6.88s    remaining: 10.4s
479:    learn: 0.3837376        test: 0.4120918 best: 0.4120779 (476)   total:
6.9s     remaining: 10.3s
480:    learn: 0.3835708        test: 0.4119252 best: 0.4119252 (480)   total:
6.91s    remaining: 10.3s
481:    learn: 0.3834717        test: 0.4119280 best: 0.4119252 (480)   total:
6.93s    remaining: 10.3s
482:    learn: 0.3834339        test: 0.4119004 best: 0.4119004 (482)   total:
```

```
6.94s    remaining: 10.3s
483:    learn: 0.3834079        test: 0.4118797 best: 0.4118797 (483)   total:
6.95s    remaining: 10.3s
484:    learn: 0.3832526        test: 0.4119045 best: 0.4118797 (483)   total:
6.97s    remaining: 10.3s
485:    learn: 0.3832507        test: 0.4119094 best: 0.4118797 (483)   total:
6.98s    remaining: 10.3s
486:    learn: 0.3830010        test: 0.4119191 best: 0.4118797 (483)   total:
7s       remaining: 10.2s
487:    learn: 0.3829774        test: 0.4118905 best: 0.4118797 (483)   total:
7.01s    remaining: 10.2s
488:    learn: 0.3828958        test: 0.4117950 best: 0.4117950 (488)   total:
7.02s    remaining: 10.2s
489:    learn: 0.3827568        test: 0.4117401 best: 0.4117401 (489)   total:
7.04s    remaining: 10.2s
490:    learn: 0.3827422        test: 0.4117454 best: 0.4117401 (489)   total:
7.04s    remaining: 10.2s
491:    learn: 0.3826548        test: 0.4116007 best: 0.4116007 (491)   total:
7.06s    remaining: 10.2s
492:    learn: 0.3826025        test: 0.4116194 best: 0.4116007 (491)   total:
7.07s    remaining: 10.1s
493:    learn: 0.3825172        test: 0.4115681 best: 0.4115681 (493)   total:
7.08s    remaining: 10.1s
494:    learn: 0.3824863        test: 0.4115388 best: 0.4115388 (494)   total:
7.09s    remaining: 10.1s
495:    learn: 0.3824636        test: 0.4115549 best: 0.4115388 (494)   total:
7.1s     remaining: 10.1s
496:    learn: 0.3824410        test: 0.4115611 best: 0.4115388 (494)   total:
7.11s    remaining: 10.1s
497:    learn: 0.3824210        test: 0.4115715 best: 0.4115388 (494)   total:
7.12s    remaining: 10s
498:    learn: 0.3824045        test: 0.4115717 best: 0.4115388 (494)   total:
7.13s    remaining: 10s
499:    learn: 0.3823747        test: 0.4115453 best: 0.4115388 (494)   total:
7.15s    remaining: 10s
500:    learn: 0.3823651        test: 0.4115648 best: 0.4115388 (494)   total:
7.16s    remaining: 9.99s
501:    learn: 0.3823357        test: 0.4115388 best: 0.4115388 (494)   total:
7.18s    remaining: 9.98s
Stopped by overfitting detector  (7 iterations wait)

bestTest = 0.4115388125
bestIteration = 494

Shrink model to first 495 iterations.
```

[454]:
```
from xgboost import XGBClassifier
cat_features = ['Title', 'FamilySize']

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [200, 300, 400, 500],
    'max_depth': [1,2,3],
    'reg_alpha': [0.001 * j for j in range(3)],   # L1 regularization
    'reg_lambda': [0.001 * j for j in range(3)],      # L2 regularization
    'learning_rate': [0.01 + j * 0.01 for j in range(5)]
}

# Initialize the XGBClassifier.
# Set tree_method='hist' to support categorical features in recent XGBoost␣
 ↪versions.
xgb = XGBClassifier(tree_method='hist', eval_metric='logloss',␣
 ↪enable_categorical = True,
                    feature_types = ['q', 'c', 'q', 'q', 'c'],
                    max_cat_to_onehot = 1)

# Set up GridSearchCV to tune the model (using 3-fold cross-validation here)
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5,␣
 ↪n_jobs=-1, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best parameters found:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
```

```
Best parameters found: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators':
200, 'reg_alpha': 0.0, 'reg_lambda': 0.0}
Best cross-validation accuracy: 0.8428472788902142
```

[464]:
```
# Now, to build the final model using early stopping, we split our training␣
 ↪data further:
X_train_1, X_val, y_train_1, y_val = train_test_split(X_train, y_train,␣
 ↪test_size=0.1, random_state=42)

# Use the best parameters found in GridSearchCV
best_params = grid_search.best_params_
xgb = XGBClassifier(n_estimators = grid_search.best_params_['n_estimators'],
                            max_depth = grid_search.best_params_['max_depth'],␣
 ↪reg_alpha = grid_search.best_params_['reg_alpha'],
                            reg_lambda = grid_search.
 ↪best_params_['reg_lambda'], learning_rate = grid_search.
 ↪best_params_['learning_rate'],
                            tree_method='hist', enable_categorical = True,␣
 ↪eval_metric='logloss',
```

```
                            feature_types = ['q', 'c', 'q', 'q', 'c'],␣
 ↪max_cat_to_onehot = 1,
                            early_stopping_rounds=10)

# Fit the model using early stopping to avoid overfitting.
xgb.fit(X_train_1, y_train_1, eval_set=[(X_val, y_val)], verbose=True)

# Optionally, predict on the test set
y_pred = xgb.predict(test)
y_pred = pd.DataFrame({'PassengerId' : test_id, 'Survived' : y_pred})
y_pred.to_csv('SurvivalPredictionsXGBoost.csv', index = False)
```

```
[0]     validation_0-logloss:0.68851
[1]     validation_0-logloss:0.68397
[2]     validation_0-logloss:0.67952
[3]     validation_0-logloss:0.67515
[4]     validation_0-logloss:0.67088
[5]     validation_0-logloss:0.66668
[6]     validation_0-logloss:0.66257
[7]     validation_0-logloss:0.65853
[8]     validation_0-logloss:0.65458
[9]     validation_0-logloss:0.65069
[10]    validation_0-logloss:0.64688
[11]    validation_0-logloss:0.64315
[12]    validation_0-logloss:0.63948
[13]    validation_0-logloss:0.63588
[14]    validation_0-logloss:0.63235
[15]    validation_0-logloss:0.62888
[16]    validation_0-logloss:0.62548
[17]    validation_0-logloss:0.62214
[18]    validation_0-logloss:0.61886
[19]    validation_0-logloss:0.61564
[20]    validation_0-logloss:0.61248
[21]    validation_0-logloss:0.60938
[22]    validation_0-logloss:0.60633
[23]    validation_0-logloss:0.60333
[24]    validation_0-logloss:0.60005
[25]    validation_0-logloss:0.59715
[26]    validation_0-logloss:0.59397
[27]    validation_0-logloss:0.59117
[28]    validation_0-logloss:0.58809
[29]    validation_0-logloss:0.58538
[30]    validation_0-logloss:0.58272
[31]    validation_0-logloss:0.57977
[32]    validation_0-logloss:0.57720
[33]    validation_0-logloss:0.57434
[34]    validation_0-logloss:0.57185
[35]    validation_0-logloss:0.56908
```

```
[36]    validation_0-logloss:0.56667
[37]    validation_0-logloss:0.56429
[38]    validation_0-logloss:0.56164
[39]    validation_0-logloss:0.55935
[40]    validation_0-logloss:0.55677
[41]    validation_0-logloss:0.55455
[42]    validation_0-logloss:0.55236
[43]    validation_0-logloss:0.54990
[44]    validation_0-logloss:0.54778
[45]    validation_0-logloss:0.54539
[46]    validation_0-logloss:0.54334
[47]    validation_0-logloss:0.54132
[48]    validation_0-logloss:0.53903
[49]    validation_0-logloss:0.53708
[50]    validation_0-logloss:0.53485
[51]    validation_0-logloss:0.53295
[52]    validation_0-logloss:0.53079
[53]    validation_0-logloss:0.52896
[54]    validation_0-logloss:0.52715
[55]    validation_0-logloss:0.52508
[56]    validation_0-logloss:0.52333
[57]    validation_0-logloss:0.52131
[58]    validation_0-logloss:0.51962
[59]    validation_0-logloss:0.51795
[60]    validation_0-logloss:0.51601
[61]    validation_0-logloss:0.51440
[62]    validation_0-logloss:0.51251
[63]    validation_0-logloss:0.51095
[64]    validation_0-logloss:0.50941
[65]    validation_0-logloss:0.50760
[66]    validation_0-logloss:0.50611
[67]    validation_0-logloss:0.50435
[68]    validation_0-logloss:0.50290
[69]    validation_0-logloss:0.50147
[70]    validation_0-logloss:0.49979
[71]    validation_0-logloss:0.49840
[72]    validation_0-logloss:0.49676
[73]    validation_0-logloss:0.49542
[74]    validation_0-logloss:0.49410
[75]    validation_0-logloss:0.49252
[76]    validation_0-logloss:0.49125
[77]    validation_0-logloss:0.48971
[78]    validation_0-logloss:0.48847
[79]    validation_0-logloss:0.48725
[80]    validation_0-logloss:0.48577
[81]    validation_0-logloss:0.48459
[82]    validation_0-logloss:0.48343
[83]    validation_0-logloss:0.48201
```

```
[84]    validation_0-logloss:0.48088
[85]    validation_0-logloss:0.47950
[86]    validation_0-logloss:0.47840
[87]    validation_0-logloss:0.47733
[88]    validation_0-logloss:0.47600
[89]    validation_0-logloss:0.47495
[90]    validation_0-logloss:0.47366
[91]    validation_0-logloss:0.47265
[92]    validation_0-logloss:0.47165
[93]    validation_0-logloss:0.47041
[94]    validation_0-logloss:0.46944
[95]    validation_0-logloss:0.46823
[96]    validation_0-logloss:0.46729
[97]    validation_0-logloss:0.46637
[98]    validation_0-logloss:0.46520
[99]    validation_0-logloss:0.46431
[100]   validation_0-logloss:0.46317
[101]   validation_0-logloss:0.46231
[102]   validation_0-logloss:0.46146
[103]   validation_0-logloss:0.46036
[104]   validation_0-logloss:0.45953
[105]   validation_0-logloss:0.45865
[106]   validation_0-logloss:0.45785
[107]   validation_0-logloss:0.45681
[108]   validation_0-logloss:0.45603
[109]   validation_0-logloss:0.45520
[110]   validation_0-logloss:0.45445
[111]   validation_0-logloss:0.45345
[112]   validation_0-logloss:0.45272
[113]   validation_0-logloss:0.45194
[114]   validation_0-logloss:0.45123
[115]   validation_0-logloss:0.45028
[116]   validation_0-logloss:0.44960
[117]   validation_0-logloss:0.44867
[118]   validation_0-logloss:0.44801
[119]   validation_0-logloss:0.44729
[120]   validation_0-logloss:0.44665
[121]   validation_0-logloss:0.44577
[122]   validation_0-logloss:0.44514
[123]   validation_0-logloss:0.44453
[124]   validation_0-logloss:0.44386
[125]   validation_0-logloss:0.44302
[126]   validation_0-logloss:0.44243
[127]   validation_0-logloss:0.44186
[128]   validation_0-logloss:0.44105
[129]   validation_0-logloss:0.44050
[130]   validation_0-logloss:0.43988
[131]   validation_0-logloss:0.43934
```

```
[132]   validation_0-logloss:0.43857
[133]   validation_0-logloss:0.43805
[134]   validation_0-logloss:0.43746
[135]   validation_0-logloss:0.43695
[136]   validation_0-logloss:0.43622
[137]   validation_0-logloss:0.43573
[138]   validation_0-logloss:0.43501
[139]   validation_0-logloss:0.43454
[140]   validation_0-logloss:0.43400
[141]   validation_0-logloss:0.43354
[142]   validation_0-logloss:0.43279
[143]   validation_0-logloss:0.43235
[144]   validation_0-logloss:0.43177
[145]   validation_0-logloss:0.43134
[146]   validation_0-logloss:0.43062
[147]   validation_0-logloss:0.43014
[148]   validation_0-logloss:0.42965
[149]   validation_0-logloss:0.42919
[150]   validation_0-logloss:0.42856
[151]   validation_0-logloss:0.42811
[152]   validation_0-logloss:0.42750
[153]   validation_0-logloss:0.42707
[154]   validation_0-logloss:0.42656
[155]   validation_0-logloss:0.42620
[156]   validation_0-logloss:0.42555
[157]   validation_0-logloss:0.42521
[158]   validation_0-logloss:0.42481
[159]   validation_0-logloss:0.42433
[160]   validation_0-logloss:0.42377
[161]   validation_0-logloss:0.42339
[162]   validation_0-logloss:0.42307
[163]   validation_0-logloss:0.42248
[164]   validation_0-logloss:0.42203
[165]   validation_0-logloss:0.42173
[166]   validation_0-logloss:0.42115
[167]   validation_0-logloss:0.42086
[168]   validation_0-logloss:0.42052
[169]   validation_0-logloss:0.42011
[170]   validation_0-logloss:0.41961
[171]   validation_0-logloss:0.41928
[172]   validation_0-logloss:0.41897
[173]   validation_0-logloss:0.41848
[174]   validation_0-logloss:0.41810
[175]   validation_0-logloss:0.41785
[176]   validation_0-logloss:0.41755
[177]   validation_0-logloss:0.41704
[178]   validation_0-logloss:0.41680
[179]   validation_0-logloss:0.41644
```

```
[180]   validation_0-logloss:0.41627
[181]   validation_0-logloss:0.41583
[182]   validation_0-logloss:0.41567
[183]   validation_0-logloss:0.41545
[184]   validation_0-logloss:0.41511
[185]   validation_0-logloss:0.41496
[186]   validation_0-logloss:0.41475
[187]   validation_0-logloss:0.41428
[188]   validation_0-logloss:0.41414
[189]   validation_0-logloss:0.41394
[190]   validation_0-logloss:0.41362
[191]   validation_0-logloss:0.41349
[192]   validation_0-logloss:0.41330
[193]   validation_0-logloss:0.41286
[194]   validation_0-logloss:0.41273
[195]   validation_0-logloss:0.41255
[196]   validation_0-logloss:0.41225
[197]   validation_0-logloss:0.41213
[198]   validation_0-logloss:0.41196
[199]   validation_0-logloss:0.41154
```

[457]:
```
X_val
```

[457]:

| | Age | Title | FamilyOutcome | Class | FamilySize |
|---|---|---|---|---|---|
| 709 | 19.658589 | Master. | 2.0 | 3.0 | Small |
| 439 | 31.000000 | Mr. | 1.0 | 2.0 | Alone |
| 840 | 20.000000 | Mr. | 1.0 | 3.0 | Alone |
| 720 | 6.000000 | Miss. | 2.0 | 2.0 | Small |
| 39 | 14.000000 | Miss. | 2.0 | 3.0 | Small |
| .. | … | … | … | … | … |
| 493 | 71.000000 | Mr. | 1.0 | 1.0 | Alone |
| 215 | 31.000000 | Miss. | 2.0 | 1.0 | Small |
| 309 | 30.000000 | Miss. | 1.0 | 1.0 | Alone |
| 822 | 38.000000 | Mr. | 1.0 | 1.0 | Alone |
| 250 | 30.900102 | Mr. | 1.0 | 3.0 | Alone |

[90 rows x 5 columns]

[ ]:
```

```