Practice Project

December 8, 2025

1 Practice Project: Titanic Survival Prediction

Estimated time needed: **30** minutes

1.1 Introduction

Now that you have a feel for how to optimize your machine learning pipeline, let's practice with a real world dataset.
You'll use cross validation and a hyperparameter grid search to optimize your machine learning pipeline.

You will use the Titanic Survival Dataset to build a classification model to predict whether a passenger survived the sinking of the Titanic, based on attributes of each passenger in the data set.

You'll start with building a Random Forest Classifier, then modify your pipeline to use a Logistic Regression estimator instead. You'll evaluate and compare your results.

This lab will help prepare you for completing the Final Project.

1.2 Objectives

After completing this lab you will be able to:

- Use scikit-learn to build a model to solve a classification problem
- Implement a pipeline to combine your preprocessing steps with a machine learning model
- Interpret the results of your modelling
- Update your pipeline with a different machine learning model
- Compare the preformances of your classifiers

1.2.1 Install the required libraries

[1]:
```
!pip install numpy
!pip install matplotlib
!pip install pandas
!pip install scikit-learn
!pip install seaborn
```

```
Collecting numpy
  Downloading
numpy-2.3.5-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata
```

```
(62 kB)
Downloading
numpy-2.3.5-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.6
MB)
                           16.6/16.6 MB
175.6 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.3.5
Collecting matplotlib
  Downloading matplotlib-3.10.7-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.61.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (113 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib) (2.3.5)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-12.0.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (8.8 kB)
Collecting pyparsing>=3 (from matplotlib)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading
matplotlib-3.10.7-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl
(8.7 MB)
                           8.7/8.7 MB
145.4 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.61.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
```

```
                        4.9/4.9 MB
153.1 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                        1.5/1.5 MB
92.1 MB/s eta 0:00:00
Downloading
pillow-12.0.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (7.0
MB)
                        7.0/7.0 MB
175.4 MB/s eta 0:00:00
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.61.0
kiwisolver-1.4.9 matplotlib-3.10.7 pillow-12.0.0 pyparsing-3.2.5
Collecting pandas
  Downloading pandas-2.3.3-cp312-cp312-
manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (91 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2.3.5)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2024.2)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading
pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (12.4
MB)
                        12.4/12.4 MB
182.3 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.3.3 tzdata-2025.2
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.22.0 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn) (2.3.5)
Collecting scipy>=1.8.0 (from scikit-learn)
  Downloading
scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(62 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
```

```
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.7.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl
(9.5 MB)
                         9.5/9.5 MB
130.9 MB/s eta 0:00:00
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading
scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7
MB)
                         35.7/35.7 MB
144.4 MB/s eta 0:00:0000:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.7.2 scipy-1.16.3
threadpoolctl-3.6.0
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/conda/lib/python3.12/site-packages (from seaborn) (2.3.5)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-
packages (from seaborn) (2.3.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/conda/lib/python3.12/site-packages (from seaborn) (3.10.7)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(4.61.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.4.9)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (12.0.0)
Requirement already satisfied: pyparsing>=3 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
```

```
packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
```

1.2.2 Import the required libraries

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
import seaborn as sns
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
```

1.2.3 Titanic Passenger data set

We'll be working with the Titanic passenger dataset to build a classification model to predict whether a passenger survied the sinking of the Titanic.
Here is the data dictionary:

Variable	Definition
survived	survived? 0 = No, 1 = yes
pclass	Ticket class (int)
sex	sex
age	age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
fare	Passenger fare
embarked	Port of Embarkation
class	Ticket class (obj)
who	man, woman, or child
adult_male	True/False

Variable	Definition
alive	yes/no
alone	yes/no

1.3 Load the Titanic dataset using Seaborn

[3]:
```
titanic = sns.load_dataset('titanic')
titanic.head()
```

[3]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

1.3.1 Select relevant features and the target

[4]:
```
titanic.count()
```

[4]:
```
survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
class          891
who            891
adult_male     891
deck           203
embark_town    889
alive          891
alone          891
dtype: int64
```

Features to drop `deck` has a lot of missing values so we'll drop it. `age` has quite a few missing values as well. Although it could be, `embarked` and `embark_town` don't seem relevant so we'll drop

them as well. It's unclear what `alive` refers to so we'll ignore it. #### Target `survived` is our target class variable.

[5]:
```
features = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'class', 'who',␣
 ↪'adult_male', 'alone']
target = 'survived'

X = titanic[features]
y = titanic[target]
```

1.3.2 Exercise 1. How balanced are the classes?

[6]:
```
y.value_counts()
```

[6]:
```
survived
0    549
1    342
Name: count, dtype: int64
```

Click here for the solution

```
y.value_counts()

So about 38% of the passengers in the data set survived.
Because of this slight imbalance, we should stratify the data when performing train/test split
```

1.3.3 Exercise 2. Split the data into training and testing sets

Don't forget to consider imbalance in the target

[8]:
```
# Enter your code here:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪stratify=y, random_state=42)
```

Click here for the solution

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, strat
```

1.3.4 Define preprocessing transformers for numerical and categorical features

Automatically detect numerical and categorical columns and assign them to separate numeric and categorical features

[9]:
```
numerical_features = X_train.select_dtypes(include=['number']).columns.tolist()
categorical_features = X_train.select_dtypes(include=['object', 'category']).
 ↪columns.tolist()
```

Define separate preprocessing pipelines for both feature types

[10]:
```
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Combine the transformers into a single column transformer We'll use the sklearn "column transformer" estimator to separately transform the features, which will then concatenate the output as a single feature space, ready for input to a machine learning estimator.

[11]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

1.3.5 Create a model pipeline

Now let's complete the model pipeline by combining the preprocessing with a Random Forest classifier

[12]:
```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

1.3.6 Define a parameter grid

We'll use the grid in a cross validation search to optimize the model

[13]:
```
param_grid = {
    'classifier__n_estimators': [50, 100],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5]
}
```

1.3.7 Perform grid search cross-validation and fit the best model to the training data

[14]:
```
# Cross-validation method
cv = StratifiedKFold(n_splits=5, shuffle=True)
```

1.3.8 Exercise 3. Train the pipeline model

[15]:
```
# Enter your code here
model = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=cv,␣
 ↪scoring='accuracy', verbose=2)
model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
```

```
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
```

```
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=50; total time=   0.1s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5,
classifier__n_estimators=100; total time=   0.2s
```

[15]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
                 SimpleImputer(strategy='median')),
                ('scaler',
                 StandardScaler())]),
['pclass',
                                                                         'age',
'sibsp',
'parch',
'fare']),
                                                                       ('cat',
Pipeline(steps=[('imputer',
                 SimpleImputer(strategy='most_frequent')),
                ('onehot',
                 OneHotEncoder(handle_unknown='ignore'))]),
                                                                        ['sex',
'class',
'who'])])),
                                        ('classifier',
```

```
 RandomForestClassifier(random_state=42))]),
              param_grid={'classifier__max_depth': [None, 10, 20],
                          'classifier__min_samples_split': [2, 5],
                          'classifier__n_estimators': [50, 100]},
              scoring='accuracy', verbose=2)
```

Click here for the solution

```
model = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=cv, scoring='accuracy', verb
model.fit(X_train, y_train)
```

1.3.9 Exercise 4. Get the model predictions from the grid search estimator on the unseen data

Also print a classification report

[16]:
```
# Enter your code here:
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.87       110
           1       0.81      0.74      0.77        69

    accuracy                           0.83       179
   macro avg       0.83      0.82      0.82       179
weighted avg       0.83      0.83      0.83       179
```

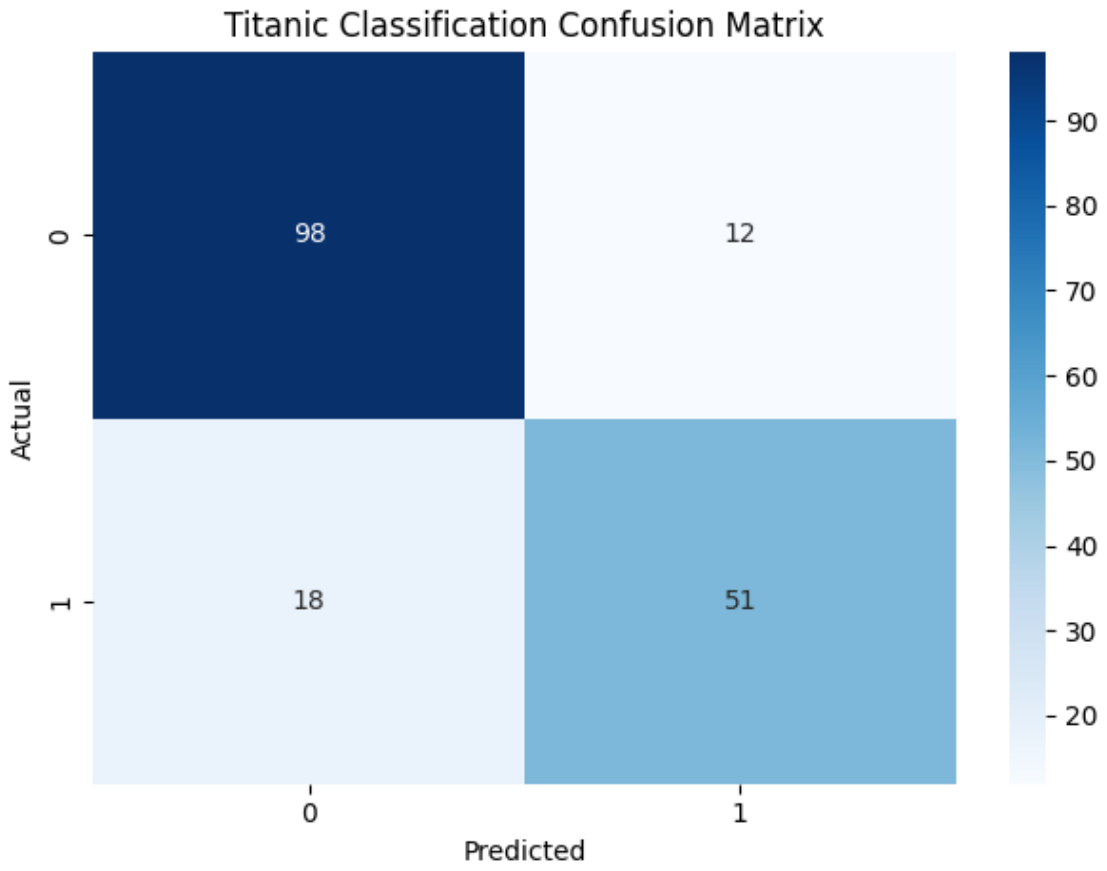
Click here for the solution

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

1.3.10 Exercise 5. Plot the confusion matrix

[17]:
```
# Enter your code here:
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')

# Set the title and labels
plt.title('Titanic Classification Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Show the plot
plt.tight_layout()
```

```
plt.show()
```



Click here for the solution

```
# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')

# Set the title and labels
plt.title('Titanic Classification Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Show the plot
plt.tight_layout()
plt.show()
```

1.4 Feature importances

Let's figure out how to get the feature importances of our overall model. You'll need to know how to do this for your final project.
First, to obtain the categorical feature importances, we have to work our way backward through the modelling pipeline to associate the feature importances with their one-hot encoded input features that were transformed from the original categorical features.

We don't need to trace back through the pipeline for the numerical features, because we didn't transfrom them into new ones in any way.
Remember, we went from categorical features to one-hot encoded features, using the 'cat' column transformer.

Here's how you trace back through the trained model to access the one-hot encoded feature names:

[ ]:
```
model.best_estimator_['preprocessor'].named_transformers_['cat'].
↪named_steps['onehot'].get_feature_names_out(categorical_features)
```

Notice how the one-hot encoded features are named - for example, `sex` was split into two boolean features indicating whether the sex is male or female.

Great! Now let's get all of the feature importances and associate them with their transformed feature names.

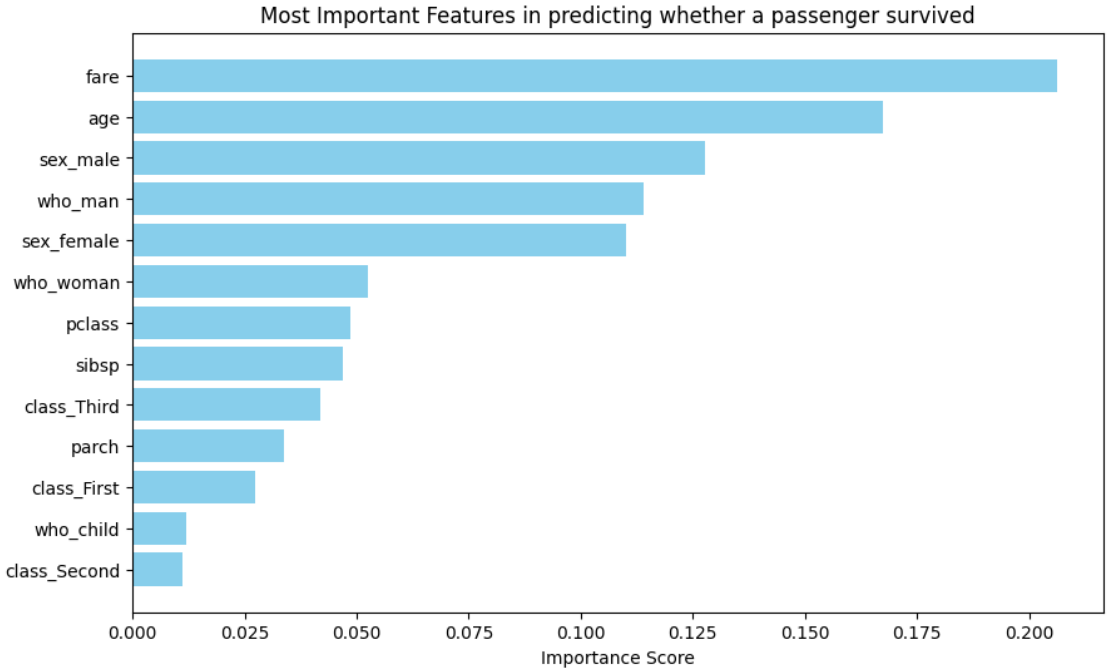
[18]:
```
feature_importances = model.best_estimator_['classifier'].feature_importances_

# Combine the numerical and one-hot encoded categorical feature names
feature_names = numerical_features + list(model.best_estimator_['preprocessor']
                                          .named_transformers_['cat']
                                          .named_steps['onehot']
                                          .
 ↪get_feature_names_out(categorical_features))
```

1.4.1 Display the feature importances in a bar plot

Define a feature importance DataFrame, then plot it

[19]:
```
importance_df = pd.DataFrame({'Feature': feature_names,
                              'Importance': feature_importances
                             }).sort_values(by='Importance', ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.gca().invert_yaxis()
plt.title('Most Important Features in predicting whether a passenger survived')
plt.xlabel('Importance Score')
plt.show()

# Print test score
```

```
test_score = model.score(X_test, y_test)
print(f"\nTest set accuracy: {test_score:.2%}")
```




```
Test set accuracy: 83.24%
```

1.4.2 Exercise 6. These are interesting results to consider.

What can you say about these feature importances? Are they informative as is?

The test set accuracy is somewhat satisfactory. However,regarding the feature impoirtances, it's crucially important to realize that there is most likely plenty of dependence amongst these variables, and a more detailed modelling approach including correlation analysis is required to draw proper conclusions. For example, no doubt there is significant information shared by the variables `age`, `sex_male`, and `who_man`. "'

1.5 Try another model

In practice you would want to try out different models and even revisit the data analysis to improve your model performance. Maybe you can engineer new features or impute missing values to be able to use more data.

With Scikit-learn's powerful pipeline class, this is easy to do in a few steps. Let's update the pipeline and the parameter grid so we can train a Logistic Regression model and compare the performance of the two models.

[20]:
```
# Replace RandomForestClassifier with LogisticRegression
pipeline.set_params(classifier=LogisticRegression(random_state=42))

# update the model's estimator to use the new pipeline
model.estimator = pipeline

# Define a new grid with Logistic Regression parameters
param_grid = {
    # 'classifier__n_estimators': [50, 100],
    # 'classifier__max_depth': [None, 10, 20],
    # 'classifier__min_samples_split': [2, 5],
    'classifier__solver' : ['liblinear'],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__class_weight' : [None, 'balanced']
}

model.param_grid = param_grid

# Fit the updated pipeline with Logistic Regression
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] END classifier__class_weight=None, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=None, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1,
```

```
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2,
classifier__solver=liblinear; total time=   0.0s
```

1.5.1 Exercise 7. Display the clasification report for the new model and compare the results to your previous model.

[21]:
```
# Enter your code here:
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86       110
           1       0.81      0.72      0.76        69

    accuracy                           0.83       179
   macro avg       0.82      0.81      0.81       179
weighted avg       0.83      0.83      0.82       179
```

Click here for the solution

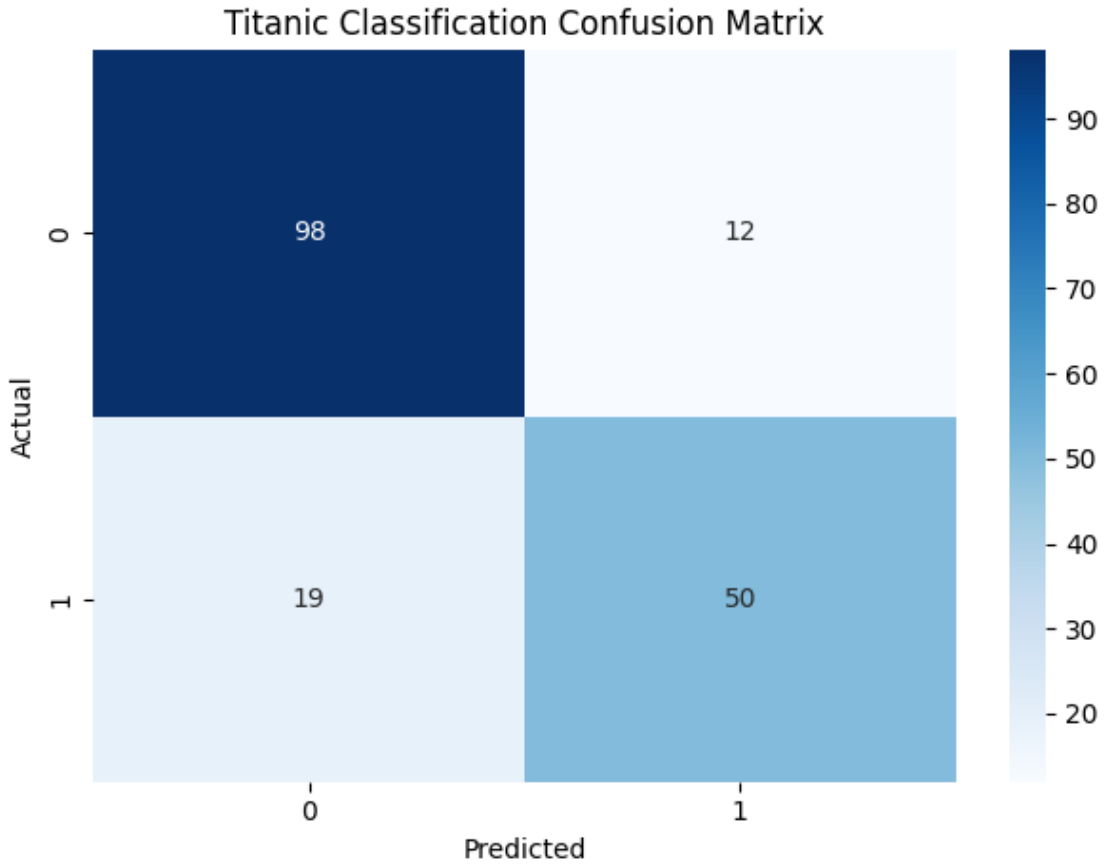
```
print(classification_report(y_test, y_pred))
```

All of the scores are slightly better for logistic regression than for random forest classification, although the differences are insignificant.

1.5.2 Exercise 8. Display the confusion matrix for the new model and compare the results to your previous model.

[23]:
```
# Enter your code here:
# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure()
```

```
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')

# Set the title and labels
plt.title('Titanic Classification Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Show the plot
plt.tight_layout()
plt.show()

# What changed in the numbers of true positives and true negatives?
```



Click here for the solution

```
# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
```

```
# Set the title and labels
plt.title('Titanic Classification Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Show the plot
plt.tight_layout()
plt.show()
```

Again, the results show a slight improvement, with one more true positive and one more true negative.

1.5.3 Extract the logistic regression feature coefficients and plot their magnitude in a bar chart.

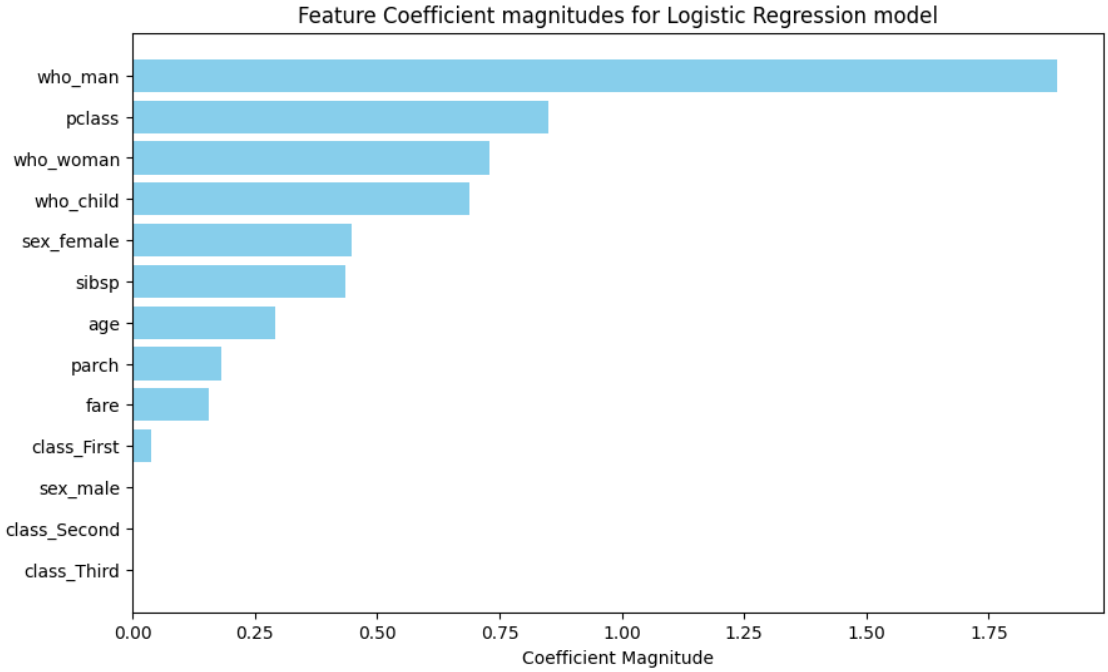
[24]:
```
coefficients = model.best_estimator_.named_steps['classifier'].coef_[0]

# Combine numerical and categorical feature names
numerical_feature_names = numerical_features
categorical_feature_names = (model.best_estimator_.named_steps['preprocessor']
                                    .named_transformers_['cat']
                                    .named_steps['onehot']
                                    .
 ↪get_feature_names_out(categorical_features)
                             )
feature_names = numerical_feature_names + list(categorical_feature_names)
```

1.5.4 Exercise 9. Plot the feature coefficient magnitudes in a bar chart

What's different about this chart than the feature importance chart for the Random Forest classifier?

[25]:
```
# Enter your code here:

# Create a DataFrame for the coefficients
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
}).sort_values(by='Coefficient', ascending=False, key=abs)  # Sort by absolute␣
 ↪values

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Coefficient'].abs(),␣
 ↪color='skyblue')
plt.gca().invert_yaxis()
plt.title('Feature Coefficient magnitudes for Logistic Regression model')
```

```
plt.xlabel('Coefficient Magnitude')
plt.show()

# Print test score
test_score = model.best_estimator_.score(X_test, y_test)
print(f"\nTest set accuracy: {test_score:.2%}")
```




```
Test set accuracy: 82.68%
```

Click here for the solution

```
# Create a DataFrame for the coefficients
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
}).sort_values(by='Coefficient', ascending=False, key=abs)  # Sort by absolute values

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Coefficient'].abs(), color='skyblue')
plt.gca().invert_yaxis()
plt.title('Feature Coefficient magnitudes for Logistic Regression model')
plt.xlabel('Coefficient Magnitude')
plt.show()
```

```
# Print test score
test_score = model.best_estimator_.score(X_test, y_test)
print(f"\nTest set accuracy: {test_score:.2%}")
```

Although the performances of the two models are virtually identical, the features that are important to the two models are very different. This suggests there must be more work to do to better grasp the actual feature importancdes. A smentioned above, it's crucially important to realize that there is most likely plenty of dependence amongst these variables, and a more detailed modelling approach including correlation analysis is required to draw proper conclusions. For example, there is significant information implied between the variables `who_man`, `who_woman`, and `who_child`, because if a person is neither a man nor a woman, then they must be a child.

1.5.5 Congratulations! You've made it this far and are now fully equipped to take on your final project!

1.6 Author

Jeff Grossman

1.6.1 Other Contributor(s)

Abhishek Gagneja