20250610_01

June 10, 2025

[29]:
```python
# Main goal : predict whether a passenger survived or not
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[21]:
```python
# Load dataset
data = sns.load_dataset('titanic')
data.head()
```

[21]:
```
   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
```

[22]:
```python
# Preview
print(data.shape)
print(data.dtypes)
```

```
(891, 15)
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
```

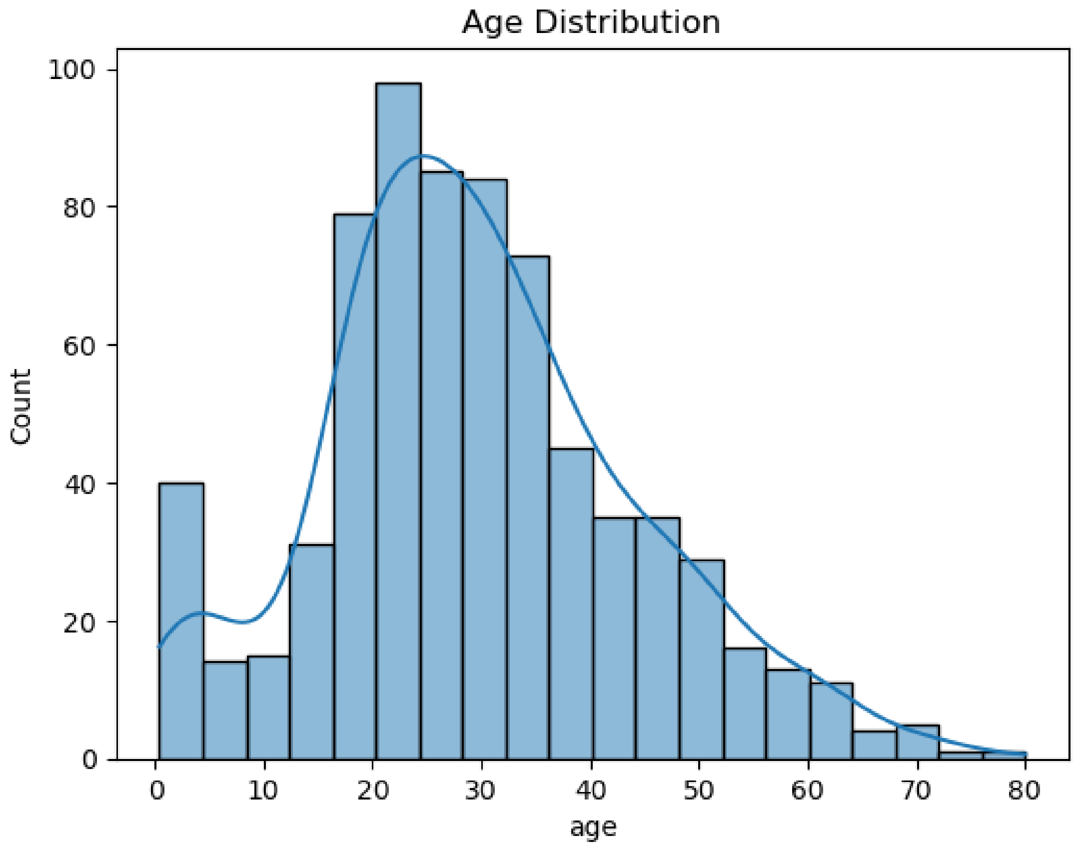
```
adult_male          bool
deck            category
embark_town       object
alive             object
alone               bool
dtype: object
```

[23]:
```python
# Check missing values
data.isnull().sum()
```

[23]:
```
survived          0
pclass            0
sex               0
age             177
sibsp             0
parch             0
fare              0
embarked          2
class             0
who               0
adult_male        0
deck            688
embark_town       2
alive             0
alone             0
dtype: int64
```

[24]:
```python
# Since deck contains too many missing values, I just choose to drop it. Yipee.
# And also embarked and embarked town are the same (I think so ?). So I just␣
 ↪decide to leave one.
data = data.drop(columns = ['deck', 'embark_town'])
```

[25]:
```python
# Take a look at the spread of age.
sns.histplot(data['age'], kde = True)
plt.title('Age Distribution')
plt.show()
```



[26]:
```
# Little bit skewed, so fill age with median
data['age'] = data['age'].fillna(data['age'].median())

# Fill embarked with mode, I mean, what else ? (Acutually, what else ? I didn't␣
↪learn that much.)
data['embarked'] = data['embarked'].fillna(data['embarked'].mode()[0])
```

[28]:
```
# Confirm
data.isnull().sum()
```

[28]:
```
survived      0
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      0
class         0
who           0
```

```
adult_male    0
alive         0
alone         0
dtype: int64
```

[31]:
```
data.head()
```

[31]:
```
   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 alive  alone
0    man        True    no  False
1  woman       False   yes  False
2  woman       False   yes   True
3  woman       False   yes  False
4    man        True    no   True
```

[32]:
```
# Our main goal is to predict survived
y = data['survived']
```

[34]:
```
# pclass and class is the same thing, we choose to keep pclass since is␣
↪numerical.
# also alive is equal to survived, so I shall drop both.
# adult_male can be define via age and sex, so we don't really need it.
# who can also be define via age and sex, drop it too.
drop_cols = ['class', 'survived', 'alive', 'adult_male', 'who']
X = data
X = X.drop(columns = drop_cols)
```

[35]:
```
X.head()
```

[35]:
```
   pclass     sex   age  sibsp  parch     fare embarked  alone
0       3    male  22.0      1      0   7.2500        S  False
1       1  female  38.0      1      0  71.2833        C  False
2       3  female  26.0      0      0   7.9250        S   True
3       1  female  35.0      1      0  53.1000        S  False
4       3    male  35.0      0      0   8.0500        S   True
```

[36]:
```
# Split time
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 42)
```

[39]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Identify column types
# Somehow we can leave Boolean alone because the model is smart enough to␣
↪handle it?
num_cols = ['pclass', 'age', 'sibsp', 'parch', 'fare']
cat_cols = ['sex', 'embarked']

# Create preprocessor
preprocessor = ColumnTransformer(transformers = [('num', StandardScaler(),␣
↪num_cols),
                                                 ('cat', OneHotEncoder(drop =␣
↪'first'), cat_cols)])

# Define pipelines
lr_pipeline = Pipeline(steps = [('preprocessor', preprocessor),
                                ('classifier', LogisticRegression(max_iter␣
↪= 1000))])

rf_pipeline = Pipeline(steps = [('preprocessor', preprocessor),
                                ('classifier',␣
↪RandomForestClassifier(random_state = 42))])
```

[41]:
```
# Time to evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix
```

[42]:
```
lr_pipeline.fit(X_train, y_train)
lr_prediction = lr_pipeline.predict(X_test)
```

[43]:
```
rf_pipeline.fit(X_train, y_train)
rf_prediction = rf_pipeline.predict(X_test)
```

[47]:
```
# Since it's a bit long, 'd better to define a function first
def print_metrics(y_true, y_prediction, model_name):
    print(f"\n{model_name} Performance:")
    print("Accuracy :", accuracy_score(y_true, y_prediction))
    print("Precision:", precision_score(y_true, y_prediction))
    print("Recall   :", recall_score(y_true, y_prediction))
    print("F1 Score :", f1_score(y_true, y_prediction))
```

[48]:
```
print_metrics(y_test, lr_prediction, "Logistic Regression")
print_metrics(y_test, rf_prediction, "Random Forest")
```

```
Logistic Regression Performance:
Accuracy : 0.8100558659217877
Precision: 0.7857142857142857
Recall   : 0.7432432432432432
F1 Score : 0.7638888888888888

Random Forest Performance:
Accuracy : 0.8212290502793296
Precision: 0.8
Recall   : 0.7567567567567568
F1 Score : 0.7777777777777778
```

[49]:

```python
# Making confusion matrix
cm = confusion_matrix(y_test, rf_prediction)

sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Random Forest Confusion Matrix")
plt.show()
```