CSCI 4521: Applied Machine Learning (Fall 2024)

Homework 3

(Due Tue, Oct. 29, 11:59 PM CT)

The RMS Titanic was a British ocean liner considered by many as "unsinkable." Unfortunately, the Titanic hit an iceberge and sank on April 15, 1912 on her trip from Southampton, England to New York City, USA. There were not enough lifeboards onboard for everyone and, as a result, an estimated 1500 people died out of the 2224 passengers and crew onboard. The Titanic disaster was one of the deadliest ship sinkings. There was a large element of luck involved in surviving the shipwreck but some people were more likely to survive than others.

rms-titanic-14047.png

In this homework, your task is to predict whether a passenger will survive the shipwreck or not. You need to use machine learning and develop classification models to accomplish this task. The only data you have available is passenger data in the dataset `titanic_dataset_csci4521.csv` which consists of the following features:

- Passenger ID,
- Ticket class (1 = first class, 2 = second class, 3 = third class),
- Passenger name,
- Sex,
- Age

Age,

- Number of siblings or spouses aboard,
- Number of parents or children aboard,
- Ticket number,
- Fare,
- Cabin number, and
- Port of Embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

and label:

- Survived ($y_i = 1$) or
- Not survived ($y_i = 0$).

You must decide if and how to clean and preprocess the data, which classification algorithms to use, which and how to tune any hyperparameters, how to measure performance, which models to select, and which final model to use.

You can use any of the coding packages we've used in class (numpy, pandas, pyspark, scikit-learn, etc.) and you must write and submit working code. Reminder, you cannot use ChatGPT or similar technologies. Please see the syllabus for more details.

You also need to submit a short report of your work describing all steps you took, explanations of why you took those steps, results, what you learned, how you might use what you learned in the future, and your conclusions. We expect the report to be well-written and clearly describe everything you've done and why.

Write your code here

```
# !apt-get install openjdk-8-jdk-headless -qq > /dev/null
# !wget -q http://archive.apache.org/dist/spark/spark-3.5.3/spark-3.5.3-bin-h
# !tar xf spark-3.5.3-bin-hadoop3.tgz
# !pip install -q findspark

# import os
# os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
# os.environ["SPARK_HOME"] = "/content/spark-3.5.3-bin-hadoop3"


import findspark
findspark.init()
from pyspark.sql import SparkSession, DataFrame
spark = SparkSession.builder.getOrCreate()
import pyspark.sql.functions  as F
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from sklearn.model_selection import  cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns


def load_data(path: str) -> DataFrame:
    df = spark.read.csv(path, header=True, inferSchema=True)
    return df

df = load_data("titanic_dataset_csci4521.csv")
df = df.drop("Name")
df = df.drop("Ticket")
df.show(30)
df.drop()
target = df.select("Survived")
features = [col for col in df.columns if col != "Survived" and col != "Passe
```

```
+-----------+--------+------+------+----+-----+-----+-------+-----------+-----
|PassengerId|Survived|Pclass|   Sex| Age|SibSp|Parch|   Fare|      Cabin|Emba
+-----------+--------+------+------+----+-----+-----+-------+-----------+-----
|          1|       0|     3|  male|22.0|    1|    0|   7.25|       NULL|
|          2|       1|     1|female|38.0|    1|    0|71.2833|        C85|
|          3|       1|     3|female|26.0|    0|    0|  7.925|       NULL|
|          4|       1|     1|female|35.0|    1|    0|   53.1|       C123|
```

```
|          4|       1|     1|female|35.0|    1|    0|   53.1|       C123|
|          5|       0|     3|  male|35.0|    0|    0|   8.05|       NULL|
|          6|       0|     3|  male|NULL|    0|    0| 8.4583|       NULL|
|          7|       0|     1|  male|54.0|    0|    0|51.8625|        E46|
|          8|       0|     3|  male| 2.0|    3|    1| 21.075|       NULL|
|          9|       1|     3|female|27.0|    0|    2|11.1333|       NULL|
|         10|       1|     2|female|14.0|    1|    0|30.0708|       NULL|
|         11|       1|     3|female| 4.0|    1|    1|   16.7|         G6|
|         12|       1|     1|female|58.0|    0|    0|  26.55|       C103|
|         13|       0|  NULL|  male|20.0| NULL|    0|   8.05|       NULL|
|         14|       0|     3|  male|39.0|    1|    5| 31.275|       NULL|
|         15|       0|     3|female|14.0|    0|    0| 7.8542|       NULL|
|         16|       1|     2|female|55.0|    0|    0|   16.0|       NULL|
|         17|       0|     3|  male| 2.0|    4|    1| 29.125|       NULL|
|         18|       1|     2|  male|NULL|    0|    0|   13.0|       NULL|
|         19|       0|     3|female|31.0|    1|    0|   18.0|       NULL|
|         20|       1|     3|female|NULL|    0|    0|  7.225|       NULL|
|         21|       0|     2|  male|35.0|    0|    0|   26.0|       NULL|
|         22|       1|     2|  male|34.0|    0|    0|   13.0|        D56|    N
|         23|       1|     3|female|15.0|    0|    0| 8.0292|       NULL|
|         24|       1|     1|  male|28.0|    0|    0|   35.5|         A6|
|         25|       0|     3|female| 8.0|  300|    1| 21.075|       NULL|
|         26|       1|     3|female|38.0|    1|    5|31.3875|       NULL|
|         27|       0|     3|  male|NULL|    0|    0|  7.225|       NULL|
|         28|       0|     1|  male|19.0|    3|    2|  263.0|C23 C25 C27|
|         29|       1|     3|female|NULL|    0|    0| 7.8792|       NULL|
|         30|       0|     3|  male|NULL|    0|    0| 7.8958|       NULL|
+-----------+--------+------+------+----+-----+-----+-------+-----------+----
only showing top 30 rows
```

```
def analyze_data(df: DataFrame,features: list) -> None:
    print(f"Number of samples: {df.count()}")
    print(f"Number of features: {len(features)}")
    for f in features:
        if f == "Cabin":
            missing_values = df.filter(F.col("Cabin").isNull())
            print(f"Number of missing cabin values: {missing_values.count()}
            df.filter(F.col("Cabin").isNotNull() & (F.col("Cabin") != "")).s
        else:
            df.select(F.mode(F.col(f)), F.mean(F.col(f)),F.stddev(F.col(f)),

analyze_data(df, features)
# 1st class cutoff: ≥ 35.5
# 2nd class cutoff: ≥ 10.5
# 3rd class cutoff: < 10.5
```

```
Number of samples: 894
Number of features: 8
+------------+-----------------+-----------------+-----------+-----------+
|mode(Pclass)|      avg(Pclass)|   stddev(Pclass)|min(Pclass)|max(Pclass)|
+------------+-----------------+-----------------+-----------+-----------+
|           3|2.308040770101925|0.8358553066015733|          1|          3|
```

```
|           3|2.3080407701019234|0.8358355066013735|           1|           3|
+------------+------------------+------------------+------------+------------+

+---------+--------+-----------+--------+--------+
|mode(Sex)|avg(Sex)|stddev(Sex)|min(Sex)|max(Sex)|
+---------+--------+-----------+--------+--------+
|     male|    NULL|       NULL|  female|    male|
+---------+--------+-----------+--------+--------+

+---------+-----------------+-----------------+--------+--------+
|mode(Age)|         avg(Age)|      stddev(Age)|min(Age)|max(Age)|
+---------+-----------------+-----------------+--------+--------+
|     24.0|30.48627094972067|23.72384742035713|   -17.0|   500.0|
+---------+-----------------+-----------------+--------+--------+

+-----------+------------------+-----------------+----------+----------+
|mode(SibSp)|        avg(SibSp)|    stddev(SibSp)|min(SibSp)|max(SibSp)|
+-----------+------------------+-----------------+----------+----------+
|          0|0.8499440089585666|10.08238999450755|        -3|       300|
+-----------+------------------+-----------------+----------+----------+

+-----------+-----------------+-----------------+----------+----------+
|mode(Parch)|       avg(Parch)|    stddev(Parch)|min(Parch)|max(Parch)|
+-----------+-----------------+-----------------+----------+----------+
|          0|2.615212527964206|66.88217006897335|         0|      2000|
+-----------+-----------------+-----------------+----------+----------+

+----------+-----------------+----------------+---------+---------+
|mode(Fare)|        avg(Fare)|    stddev(Fare)|min(Fare)|max(Fare)|
+----------+-----------------+----------------+---------+---------+
|      8.05|32.18915838926171|49.6250740726904|      0.0| 512.3292|
+----------+-----------------+----------------+---------+---------+

Number of missing cabin values: 690
+-----------+--------+------+
|      Cabin|    Fare|Pclass|
+-----------+--------+------+
|        C85| 71.2833|     1|
|       C123|    53.1|     1|
|        E46| 51.8625|     1|
|         G6|    16.7|     3|
|       C103|   26.55|     1|
|        D56|    13.0|     2|
|         A6|    35.5|     1|
|C23 C25 C27|   263.0|     1|
|        B78|146.5208|     1|
|        D33| 76.7292|     1|
|        B30| 61.9792|     1|
|        C52|    35.5|     1|
|        B28|    80.0|     1|
|        C83|  83.475|     1|
|        F33|    10.5|     2|
|      F G73|    7.65|     3|
```

```
def replace_missing(df: DataFrame, features: list) -> DataFrame:
```

```
    for f in features:

        if f == "Age":
            resonable_ages = df.filter((F.col(f) >= 0) & (F.col(f) <= 100))
            avg_age = resonable_ages.select(F.mean(F.col(f))).collect()[0][0
            df = df.withColumn(f, F.when(F.col(f).isNull(), avg_age).otherwi
        elif f == "Cabin":
            df = df.withColumn(f,
                               F.when(F.col("Fare") >= 35.5, "A").when((F.co
                                                                       "B")
        elif f == "Pclass":
            df = df.withColumn(f, F.when(F.col("Fare") >= 35.5, 1).when((F.c
                                                                       2).o
        else:
            mode = df.select(F.mode(F.col(f))).collect()[0][0]
            df = df.withColumn(f, F.when(F.col(f).isNull(), mode).otherwise(
    return df

df = replace_missing(df, features)
df.show(30)
```

```
+-----------+--------+------+------+------------------+-----+-----+-------+---
|PassengerId|Survived|Pclass|   Sex|               Age|SibSp|Parch|   Fare|Cal
+-----------+--------+------+------+------------------+-----+-----+-------+---
|          1|       0|     3|  male|              22.0|    1|    0|   7.25|
|          2|       1|     1|female|              38.0|    1|    0|71.2833|
|          3|       1|     3|female|              26.0|    0|    0|  7.925|
|          4|       1|     1|female|              35.0|    1|    0|   53.1|
|          5|       0|     3|  male|              35.0|    0|    0|   8.05|
|          6|       0|     3|  male|29.702485955056176|    0|    0| 8.4583|
|          7|       0|     1|  male|              54.0|    0|    0|51.8625|
|          8|       0|     2|  male|               2.0|    3|    1| 21.075|
|          9|       1|     2|female|              27.0|    0|    2|11.1333|
|         10|       1|     2|female|              14.0|    1|    0|30.0708|
|         11|       1|     2|female|               4.0|    1|    1|   16.7|
|         12|       1|     2|female|              58.0|    0|    0|  26.55|
|         13|       0|     3|  male|              20.0|    0|    0|   8.05|
|         14|       0|     2|  male|              39.0|    1|    5| 31.275|
|         15|       0|     3|female|              14.0|    0|    0| 7.8542|
|         16|       1|     2|female|              55.0|    0|    0|   16.0|
|         17|       0|     2|  male|               2.0|    4|    1| 29.125|
|         18|       1|     2|  male|29.702485955056176|    0|    0|   13.0|
|         19|       0|     2|female|              31.0|    1|    0|   18.0|
|         20|       1|     3|female|29.702485955056176|    0|    0|  7.225|
|         21|       0|     2|  male|              35.0|    0|    0|   26.0|
|         22|       1|     2|  male|              34.0|    0|    0|   13.0|
|         23|       1|     3|female|              15.0|    0|    0| 8.0292|
|         24|       1|     1|  male|              28.0|    0|    0|   35.5|
|         25|       0|     2|female|               8.0|  300|    1| 21.075|
|         26|       1|     2|female|              38.0|    1|    5|31.3875|
|         27|       0|     3|  male|29.702485955056176|    0|    0|  7.225|
```

```
|          28|       0|     1|  male|              19.0|    3|    2|  263.0|
|          29|       1|     3|female|29.702485955056176|    0|    0| 7.8792|
|          30|       0|     3|  male|29.702485955056176|    0|    0| 7.8958|
+------------+--------+------+------+------------------+-----+-----+-------+--
only showing top 30 rows
```

```
print(f"Before removing duplicates:{df.count()}")
def remove_duplicates(df: DataFrame):
    duplicates = df.groupBy("PassengerId").count().where(F.col("count") > 1)
    duplicates.show()
    df = df.dropDuplicates(["PassengerId"])
    return df

df = remove_duplicates(df)
df.show(30)
print(f"After removing duplicates:{df.count()}")
```

```
Before removing duplicates:894
+------------+-----+
|PassengerId|count|
+------------+-----+
|         164|    2|
|         510|    2|
|         334|    2|
+------------+-----+

+------------+--------+------+------+------------------+-----+-----+-------+--
|PassengerId|Survived|Pclass|   Sex|               Age|SibSp|Parch|   Fare|Cal
+------------+--------+------+------+------------------+-----+-----+-------+--
|           1|       0|     3|  male|              22.0|    1|    0|   7.25|
|           2|       1|     1|female|              38.0|    1|    0|71.2833|
|           3|       1|     3|female|              26.0|    0|    0|  7.925|
|           4|       1|     1|female|              35.0|    1|    0|   53.1|
|           5|       0|     3|  male|              35.0|    0|    0|   8.05|
|           6|       0|     3|  male|29.702485955056176|    0|    0| 8.4583|
|           7|       0|     1|  male|              54.0|    0|    0|51.8625|
|           8|       0|     2|  male|               2.0|    3|    1| 21.075|
|           9|       1|     2|female|              27.0|    0|    2|11.1333|
|          10|       1|     2|female|              14.0|    1|    0|30.0708|
|          11|       1|     2|female|               4.0|    1|    1|   16.7|
|          12|       1|     2|female|              58.0|    0|    0|  26.55|
|          13|       0|     3|  male|              20.0|    0|    0|   8.05|
|          14|       0|     2|  male|              39.0|    1|    5| 31.275|
|          15|       0|     3|female|              14.0|    0|    0| 7.8542|
|          16|       1|     2|female|              55.0|    0|    0|   16.0|
|          17|       0|     2|  male|               2.0|    4|    1| 29.125|
|          18|       1|     2|  male|29.702485955056176|    0|    0|   13.0|
|          19|       0|     2|female|              31.0|    1|    0|   18.0|
|          20|       1|     3|female|29.702485955056176|    0|    0|  7.225|
|          21|       0|     2|  male|              35.0|    0|    0|   26.0|
|          22|       1|     2|  male|              34.0|    0|    0|   13.0|
|          23|       1|     3|female|              15.0|    0|    0| 8.0292|
```

```
|         24|       1|     1|  male|              28.0|    0|    0|   35.5|
|         25|       0|     2|female|               8.0|  300|    1| 21.075|
|         26|       1|     2|female|              38.0|    1|    5|31.3875|
|         27|       0|     3|  male|29.702485955056176|    0|    0|  7.225|
|         28|       0|     1|  male|              19.0|    3|    2|  263.0|
|         29|       1|     3|female|29.702485955056176|    0|    0| 7.8792|
|         30|       0|     3|  male|29.702485955056176|    0|    0| 7.8958|
+-----------+--------+------+------+------------------+-----+-----+-------+--
only showing top 30 rows

After removing duplicates:891
```

```
analyze_data(df,features)
```

```
Number of samples: 891
Number of features: 8
+------------+------------------+-----------------+-----------+-----------+
|mode(Pclass)|       avg(Pclass)|   stddev(Pclass)|min(Pclass)|max(Pclass)|
+------------+------------------+-----------------+-----------+-----------+
|           2|2.1582491582491583|0.760460289862104|          1|          3|
+------------+------------------+-----------------+-----------+-----------+

+---------+--------+-----------+--------+--------+
|mode(Sex)|avg(Sex)|stddev(Sex)|min(Sex)|max(Sex)|
+---------+--------+-----------+--------+--------+
|     male|    NULL|       NULL|  female|    male|
+---------+--------+-----------+--------+--------+

+------------------+-----------------+------------------+--------+--------+
|         mode(Age)|         avg(Age)|       stddev(Age)|min(Age)|max(Age)|
+------------------+-----------------+------------------+--------+--------+
|29.702485955056176|30.36611952861944|21.255576857980408|   -17.0|   500.0|
+------------------+-----------------+------------------+--------+--------+

+-----------+------------------+------------------+----------+----------+
|mode(SibSp)|        avg(SibSp)|     stddev(SibSp)|min(SibSp)|max(SibSp)|
+-----------+------------------+------------------+----------+----------+
|          0|0.8496071829405163|10.093598336253093|        -3|       300|
+-----------+------------------+------------------+----------+----------+

+-----------+------------------+-----------------+----------+----------+
|mode(Parch)|        avg(Parch)|    stddev(Parch)|min(Parch)|max(Parch)|
+-----------+------------------+-----------------+----------+----------+
|          0|2.6240179573512905|66.99462535893419|         0|      2000|
+-----------+------------------+-----------------+----------+----------+

+----------+-----------------+-----------------+---------+---------+
|mode(Fare)|        avg(Fare)|     stddev(Fare)|min(Fare)|max(Fare)|
+----------+-----------------+-----------------+---------+---------+
|      8.05|32.20420796857457|49.69342859718088|      0.0| 512.3292|
+----------+-----------------+-----------------+---------+---------+

Number of missing cabin values: 0
+-----+-------+------+
|Cabin|   Fare|Pclass|
```

```
|Cabin|   Fare|Pclass|
+-----+-------+------+
|    C|   7.25|     3|
|    A|71.2833|     1|
|    C|  7.925|     3|
|    A|   53.1|     1|
|    C|   8.05|     3|
|    C| 8.4583|     3|
|    A|51.8625|     1|
|    B| 21.075|     2|
|    B|11.1333|     2|
|    B|30.0708|     2|
|    B|   16.7|     2|
|    B|  26.55|     2|
|    C|   8.05|     3|
|    B| 31.275|     2|
|    C| 7.8542|     3|
|    B|   16.0|     2|
```

```
print(f'Before removing outliers: {df.count()}')
def remove_outliers(df: DataFrame, features: list) -> DataFrame:
    result = df

    for f in features:
        if f not in ["Age", "Fare", "SibSp", "Parch"]:
            continue

        if f == "Age":
            result = result.filter((F.col(f) >= 0) & (F.col(f) <= 100))

        elif f == "Fare":
            result = result.filter(F.col(f) <= 350)

        elif f == "SibSp" or f == "Parch":
            result = result.filter((F.col(f) >= 0) & (F.col(f) <= 10))

        avg_val = result.select(F.mean(F.col(f))).collect()[0][0]
        std_val = result.select(F.stddev(F.col(f))).collect()[0][0]

        if std_val is None or avg_val is None:
            continue
        cut_off = 3 * std_val
        lower = avg_val - cut_off
        upper = avg_val + cut_off
        result = result.filter(F.col(f).between(lower, upper))

    return result

df = remove_outliers(df, features)
target = df.select("Survived")
print(f"After removing outliers: {df.count()}")
```

```
print(f"After removing outliers: {df.count()}")
df.show(30)
```

```
Before removing outliers: 891
After removing outliers: 808
+-----------+--------+------+------+-----------------+-----+-----+--------+-
|PassengerId|Survived|Pclass|   Sex|              Age|SibSp|Parch|    Fare|Ca
+-----------+--------+------+------+-----------------+-----+-----+--------+-
|          1|       0|     3|  male|             22.0|    1|    0|    7.25|
|          2|       1|     1|female|             38.0|    1|    0| 71.2833|
|          3|       1|     3|female|             26.0|    0|    0|   7.925|
|          4|       1|     1|female|             35.0|    1|    0|    53.1|
|          5|       0|     3|  male|             35.0|    0|    0|    8.05|
|          6|       0|     3|  male|29.702485955056176|    0|    0|  8.4583|
|          7|       0|     1|  male|             54.0|    0|    0| 51.8625|
|          8|       0|     2|  male|              2.0|    3|    1|  21.075|
|          9|       1|     2|female|             27.0|    0|    2| 11.1333|
|         10|       1|     2|female|             14.0|    1|    0| 30.0708|
|         11|       1|     2|female|              4.0|    1|    1|    16.7|
|         12|       1|     2|female|             58.0|    0|    0|   26.55|
|         13|       0|     3|  male|             20.0|    0|    0|    8.05|
|         15|       0|     3|female|             14.0|    0|    0|  7.8542|
|         16|       1|     2|female|             55.0|    0|    0|    16.0|
|         18|       1|     2|  male|29.702485955056176|    0|    0|    13.0|
|         19|       0|     2|female|             31.0|    1|    0|    18.0|
|         20|       1|     3|female|29.702485955056176|    0|    0|   7.225|
|         21|       0|     2|  male|             35.0|    0|    0|    26.0|
|         22|       1|     2|  male|             34.0|    0|    0|    13.0|
|         23|       1|     3|female|             15.0|    0|    0|  8.0292|
|         24|       1|     1|  male|             28.0|    0|    0|    35.5|
|         27|       0|     3|  male|29.702485955056176|    0|    0|   7.225|
|         29|       1|     3|female|29.702485955056176|    0|    0|  7.8792|
|         30|       0|     3|  male|29.702485955056176|    0|    0|  7.8958|
|         31|       0|     2|  male|             40.0|    0|    0| 27.7208|
|         32|       1|     1|female|29.702485955056176|    1|    0|146.5208|
|         33|       1|     3|female|29.702485955056176|    0|    0|    7.75|
|         34|       0|     2|  male|             66.0|    0|    0|    10.5|
|         35|       0|     1|  male|             28.0|    1|    0| 82.1708|
+-----------+--------+------+------+-----------------+-----+-----+--------+-
only showing top 30 rows
```

```
analyze_data(df, features)
```

```
Number of samples: 808
Number of features: 8
+------------+----------------+------------------+-----------+-----------+
|mode(Pclass)|     avg(Pclass)|    stddev(Pclass)|min(Pclass)|max(Pclass)|
+------------+----------------+------------------+-----------+-----------+
|           3|2.23019801980198|0.7393885148264666|          1|          3|
+------------+----------------+------------------+-----------+-----------+

+---------+--------+-----------+--------+--------+
|mode(Sex)|avg(Sex)|stddev(Sex)|min(Sex)|max(Sex)|
+---------+--------+-----------+--------+--------+
```

```
+--------+-------+----------+-------+-------+
|    male|   NULL|      NULL| female|   male|
+--------+-------+----------+-------+-------+

+-----------------+---------------+-----------------+-------+-------+
|        mode(Age)|       avg(Age)|      stddev(Age)|min(Age)|max(Age)|
+-----------------+---------------+-----------------+-------+-------+
|29.702485955056176|29.8433046118867|12.022755391076906|   0.42|   66.0|
+-----------------+---------------+-----------------+-------+-------+

+----------+-----------------+-----------------+---------+---------+
|mode(SibSp)|       avg(SibSp)|    stddev(SibSp)|min(SibSp)|max(SibSp)|
+----------+-----------------+-----------------+---------+---------+
|         0|0.3378712871287129|0.6002489555936588|        0|        3|
+----------+-----------------+-----------------+---------+---------+

+----------+------------------+-----------------+---------+---------+
|mode(Parch)|        avg(Parch)|    stddev(Parch)|min(Parch)|max(Parch)|
+----------+------------------+-----------------+---------+---------+
|         0|0.24504950495049505|0.5560082974114121|        0|        2|
+----------+------------------+-----------------+---------+---------+

+---------+----------------+-----------------+--------+--------+
|mode(Fare)|       avg(Fare)|     stddev(Fare)|min(Fare)|max(Fare)|
+---------+----------------+-----------------+--------+--------+
|     8.05|24.75664084158412|27.628765202026557|     0.0|  151.55|
+---------+----------------+-----------------+--------+--------+

Number of missing cabin values: 0
+-----+-------+------+
|Cabin|   Fare|Pclass|
+-----+-------+------+
|    C|   7.25|     3|
|    A|71.2833|     1|
|    C|  7.925|     3|
|    A|   53.1|     1|
|    C|   8.05|     3|
|    C| 8.4583|     3|
|    A|51.8625|     1|
|    B| 21.075|     2|
|    B|11.1333|     2|
|    B|30.0708|     2|
|    B|   16.7|     2|
|    B|  26.55|     2|
|    C|   8.05|     3|
|    C| 7.8542|     3|
|    B|   16.0|     2|
|    B|   13.0|     2|
```

```
def split_data(df: DataFrame) -> tuple:
    train,test = df.randomSplit([0.8, 0.2], seed=42)
    return train, test

train_df, test_df = split_data(df)
print(f"Train size: {train_df.count()}")
```

```
print(f"Train size: {train_df.count()}")
print(f"Test size: {test_df.count()}")
```

```
Train size: 678
Test size: 130
```

```
def encode_and_standardize(train_df: DataFrame, test_df: DataFrame) -> tuple
    cat_features = ["Sex", "Embarked", "Cabin"]
    for f in cat_features:
        indexer = StringIndexer(inputCol=f, outputCol=f + "_indexed")
        train_df = indexer.fit(train_df).transform(train_df)
        test_df = indexer.fit(test_df).transform(test_df)
        encoder = OneHotEncoder(inputCol=f + "_indexed", outputCol=f + "_enc
        train_df = encoder.fit(train_df).transform(train_df)
        test_df = encoder.fit(test_df).transform(test_df)

    indexer = StringIndexer(inputCol="Survived", outputCol="Label")
    train_df = indexer.fit(train_df).transform(train_df)
    test_df = indexer.fit(test_df).transform(test_df)

    numeric_features = ["Pclass", "Age", "SibSp", "Parch", "Fare"]
    cat_features_encoded = ["Sex_encoded", "Embarked_encoded", "Cabin_encode

    assembler = VectorAssembler(inputCols=numeric_features, outputCol="numer
    train_df = assembler.transform(train_df)
    test_df = assembler.transform(test_df)

    scaler = StandardScaler(inputCol="numeric_features", outputCol="scaled_f
    train_df = scaler.fit(train_df).transform(train_df)
    test_df = scaler.fit(test_df).transform(test_df)

    assembler = VectorAssembler(inputCols=cat_features_encoded + ["scaled_fe
    train_df = assembler.transform(train_df)
    test_df = assembler.transform(test_df)

    train_df = train_df.select("features", "Label")
    test_df = test_df.select("features", "Label")

    return train_df, test_df

train_df, test_df = encode_and_standardize(train_df, test_df)
train_pd = train_df.toPandas()
test_pd = test_df.toPandas()
train_pd.head()
```

	features	Label
0	[1.0, 1.0, 0.0, 1.0, 0.0, 1.0356338417914215, ...	0.0

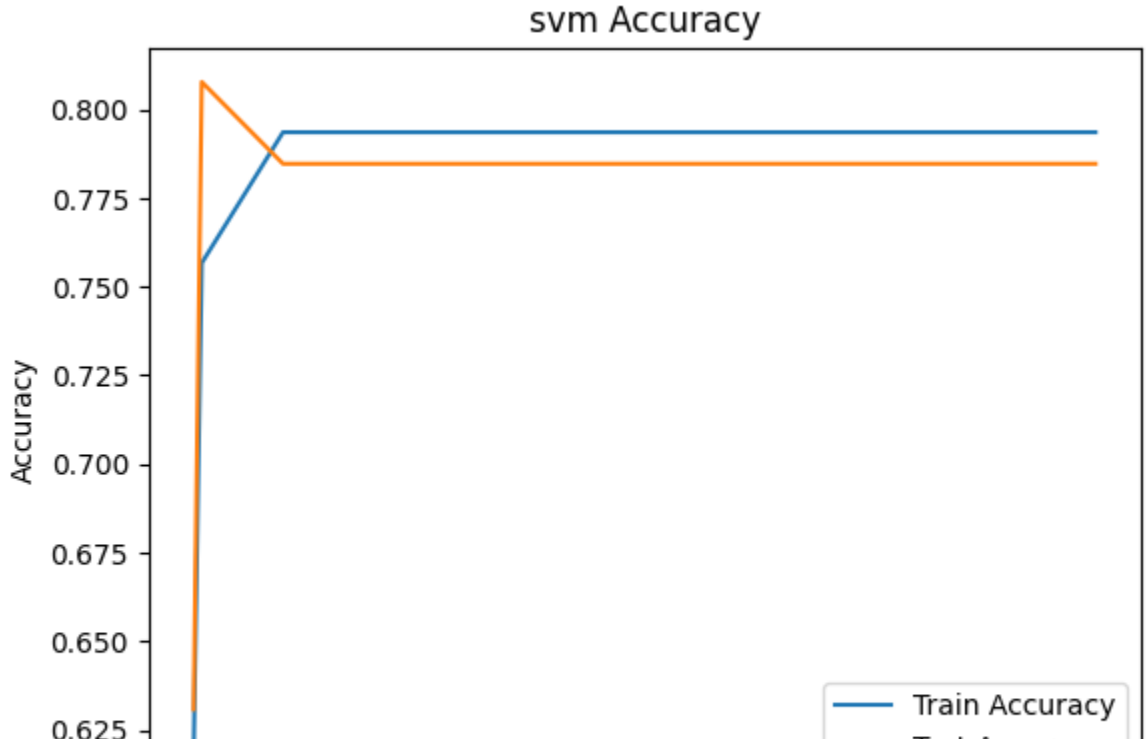
1	[0.0, 0.0, 1.0, 0.0, 0.0, -1.639260423864141, ...	1.0
2	[0.0, 1.0, 0.0, 0.0, 0.0, -1.639260423864141, ...	1.0
3	[1.0, 1.0, 0.0, 1.0, 0.0, 1.0356338417914215, ...	0.0
4	[1.0, 0.0, 0.0, 1.0, 0.0, 1.0356338417914215, ...	0.0

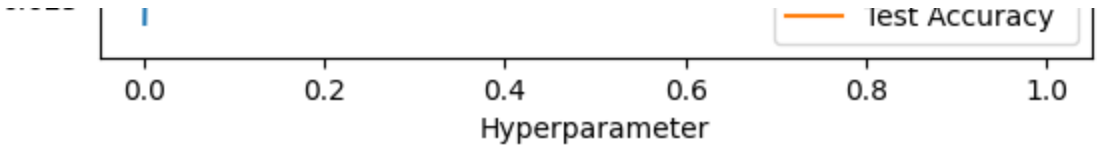
Next steps: View recommended plots New interactive sheet

```
def plot_accuracy(hyperparameters, train_accuracy, test_accuracy, model_name
    plt.plot(hyperparameters, train_accuracy, label="Train Accuracy")
    plt.plot(hyperparameters, test_accuracy, label="Test Accuracy")
    plt.xlabel("Hyperparameter")
    plt.ylabel("Accuracy")
    plt.title(f"{model_name} Accuracy")
    plt.legend()

def create_confusion_matrix(y_true, y_pred, model):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True)
    plt.title(f"{model} Confusion Matrix")
    plt.xlabel("Predicted y")
    plt.ylabel("Actual y")


def svm_model(train_pd: pd.DataFrame, test_pd: pd.DataFrame) -> tuple:
    X_train = train_pd["features"].tolist()
    X_test = test_pd["features"].tolist()
    y_train = train_pd["Label"].tolist()
    y_test = test_pd["Label"].tolist()
    test_accuracy = []
    train_accuracy = []
    best_model = None
    best_score = 0

    for c in [0.001,0.01,0.1,1]:
        model = svm.SVC(C=c, kernel="linear",random_state=42)
        scores = cross_val_score(model, X_train, y_train, cv=5)
        print(f"C={c}, avg accuracy: {scores.mean()}")
        model.fit(X_train, y_train)

        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)
```

```
        train_acc = accuracy_score(y_train, train_pred)
        test_acc = accuracy_score(y_test, test_pred)
        print(f"C={c}, train accuracy: {train_acc}")
        print(f"C={c}, test accuracy: {test_acc}")
        train_accuracy.append(train_acc)
        test_accuracy.append(test_acc)

        if test_acc > best_score:
            best_score = test_acc
            best_model = model
    best_pred = best_model.predict(X_test)

    return best_pred, train_accuracy, test_accuracy, y_test


y_pred, train_accuracy, test_accuracy, y_test = svm_model(train_pd, test_pd)
plot_accuracy([0.001,0.01,0.1,1], train_accuracy, test_accuracy, "svm")
```

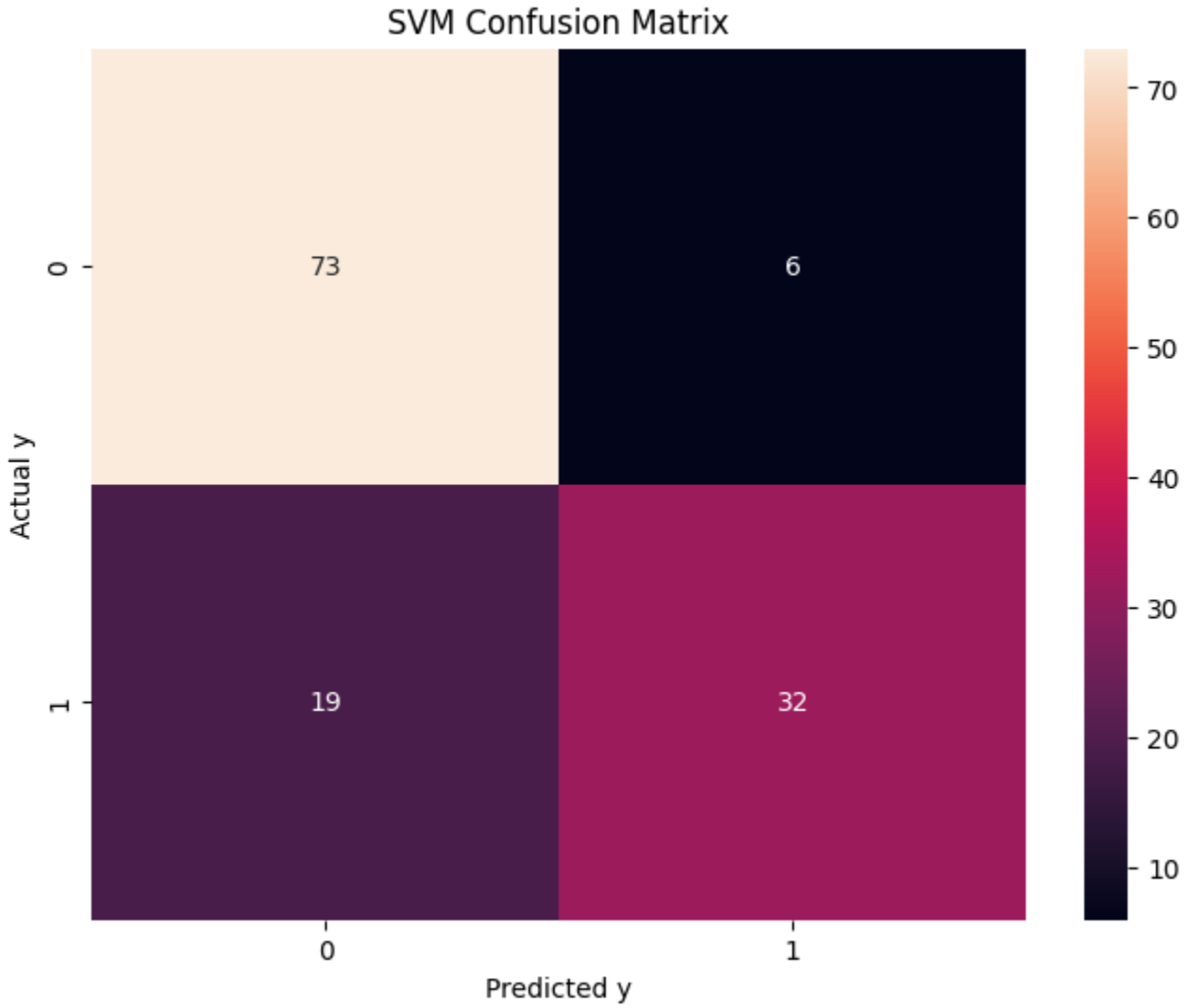
```
C=0.001, avg accuracy: 0.613562091503268
C=0.001, train accuracy: 0.6179941002949852
C=0.001, test accuracy: 0.6307692307692307
C=0.01, avg accuracy: 0.7330501089324619
C=0.01, train accuracy: 0.7566371681415929
C=0.01, test accuracy: 0.8076923076923077
C=0.1, avg accuracy: 0.7934967320261438
C=0.1, train accuracy: 0.7935103244837758
C=0.1, test accuracy: 0.7846153846153846
C=1, avg accuracy: 0.7934967320261438
C=1, train accuracy: 0.7935103244837758
C=1, test accuracy: 0.7846153846153846
```






```
print(f"Best SVM accuracy: {accuracy_score(y_test, y_pred)}")
create_confusion_matrix(y_test, y_pred, "SVM")
# 73 passengers correctly classified as survived
# 32 passengers correctly classified as not survived
# 6 passengers incorrectly classified as survived
# 19 passengers incorrectly classified as not survived
```

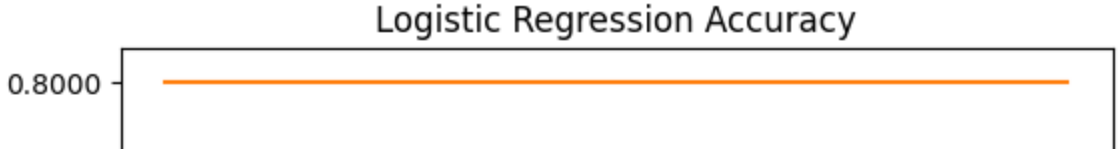
```
Best SVM accuracy: 0.8076923076923077
```

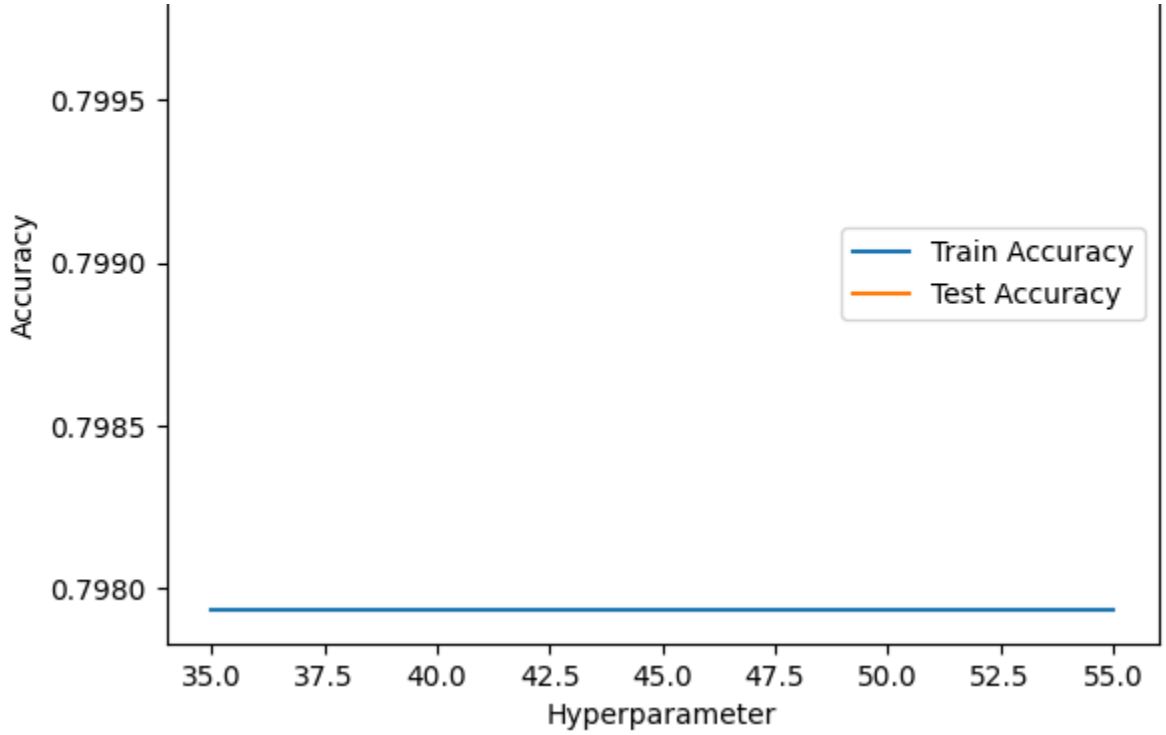



```
def lr_model(train_pd: pd.DataFrame, test_pd: pd.DataFrame) -> tuple:
    X_train = train_pd["features"].tolist()
    X_test = test_pd["features"].tolist()
    y_train = train_pd["Label"].tolist()
    y_test = test_pd["Label"].tolist()
    test_accuracy = []
```

```
    train_accuracy = []
    best_model = None
    best_score = 0

    for max_iter in [35,40,45,50,55]:
        model = LogisticRegression(max_iter=max_iter, random_state=42)
        scores = cross_val_score(model, X_train, y_train, cv=5)
        print(f"max_iter={max_iter}, avg accuracy: {scores.mean()}")
        model.fit(X_train, y_train)

        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)

        train_acc = accuracy_score(y_train, train_pred)
        test_acc = accuracy_score(y_test, test_pred)
        print(f"max_iter={max_iter}, train accuracy: {train_acc}")
        print(f"max_iter={max_iter}, test accuracy: {test_acc}")
        train_accuracy.append(train_acc)
        test_accuracy.append(test_acc)

        if test_acc > best_score:
            best_score = test_acc
            best_model = model
    best_pred = best_model.predict(X_test)

    return best_pred, train_accuracy, test_accuracy, y_test

y_pred, train_accuracy, test_accuracy, y_test = lr_model(train_pd, test_pd)
plot_accuracy([35,40,45,50,55], train_accuracy, test_accuracy, "Logistic Reg
```

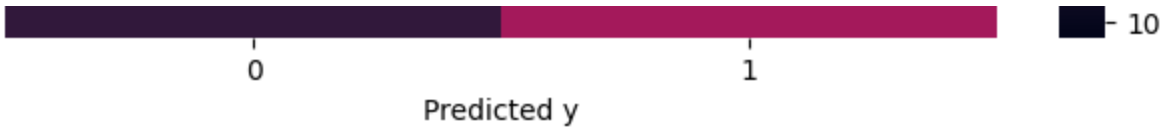
```
max_iter=35, avg accuracy: 0.7949564270152505
max_iter=35, train accuracy: 0.7979351032448377
max_iter=35, test accuracy: 0.8
max_iter=40, avg accuracy: 0.7949564270152505
max_iter=40, train accuracy: 0.7979351032448377
max_iter=40, test accuracy: 0.8
max_iter=45, avg accuracy: 0.7949564270152505
max_iter=45, train accuracy: 0.7979351032448377
max_iter=45, test accuracy: 0.8
max_iter=50, avg accuracy: 0.7949564270152505
max_iter=50, train accuracy: 0.7979351032448377
max_iter=50, test accuracy: 0.8
max_iter=55, avg accuracy: 0.7949564270152505
max_iter=55, train accuracy: 0.7979351032448377
max_iter=55, test accuracy: 0.8
```






```
print(f"Best Logistic Regression accuracy: {accuracy_score(y_test, y_pred)}"
create_confusion_matrix(y_test, y_pred, "Logistic Regression")
```

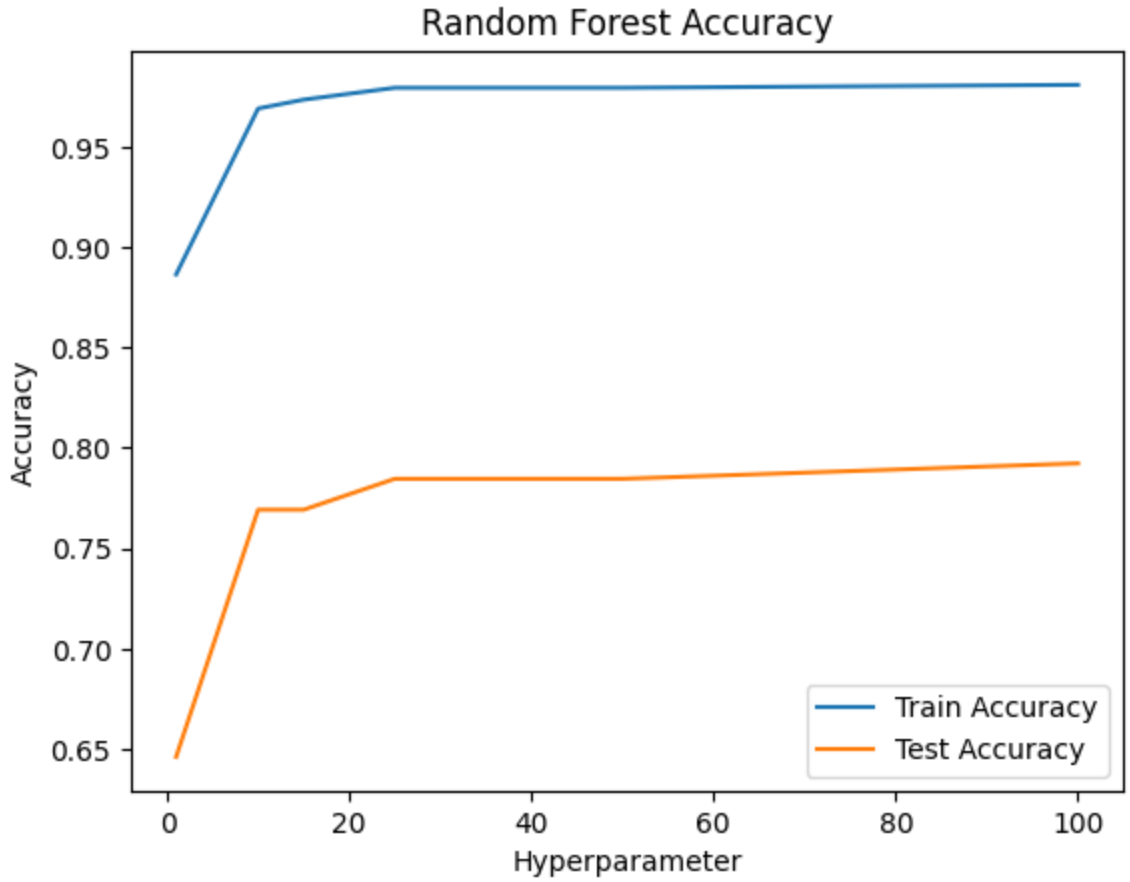
```
Best Logistic Regression accuracy: 0.8
```




```
def rf_model(train_pd: pd.DataFrame, test_pd: pd.DataFrame) -> tuple:
    X_train = train_pd["features"].tolist()
    X_test = test_pd["features"].tolist()
    y_train = train_pd["Label"].tolist()
    y_test = test_pd["Label"].tolist()
    test_accuracy = []
    train_accuracy = []
    best_model = None
    best_score = 0

    for n in [1,10,15,25,50,100]:
        model = RandomForestClassifier(n_estimators=n, random_state=42)
        scores = cross_val_score(model, X_train, y_train, cv=5)
        print(f"n_estimators={n}, avg accuracy: {scores.mean()}")
        model.fit(X_train, y_train)

        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)

        train_acc = accuracy_score(y_train, train_pred)
        test_acc = accuracy_score(y_test, test_pred)
        print(f"n_estimators={n}, train accuracy: {train_acc}")
        print(f"n_estimators={n}, test accuracy: {test_acc}")
        train_accuracy.append(train_acc)
        test_accuracy.append(test_acc)

        if test_acc > best_score:
            best_score = test_acc
            best_model = model
    best_pred = best_model.predict(X_test)

    return best_pred, train_accuracy, test_accuracy, y_test

y_pred, train_accuracy, test_accuracy, y_test = rf_model(train_pd, test_pd)
plot_accuracy([1,10,15,25,50,100], train_accuracy, test_accuracy, "Random Fo
```

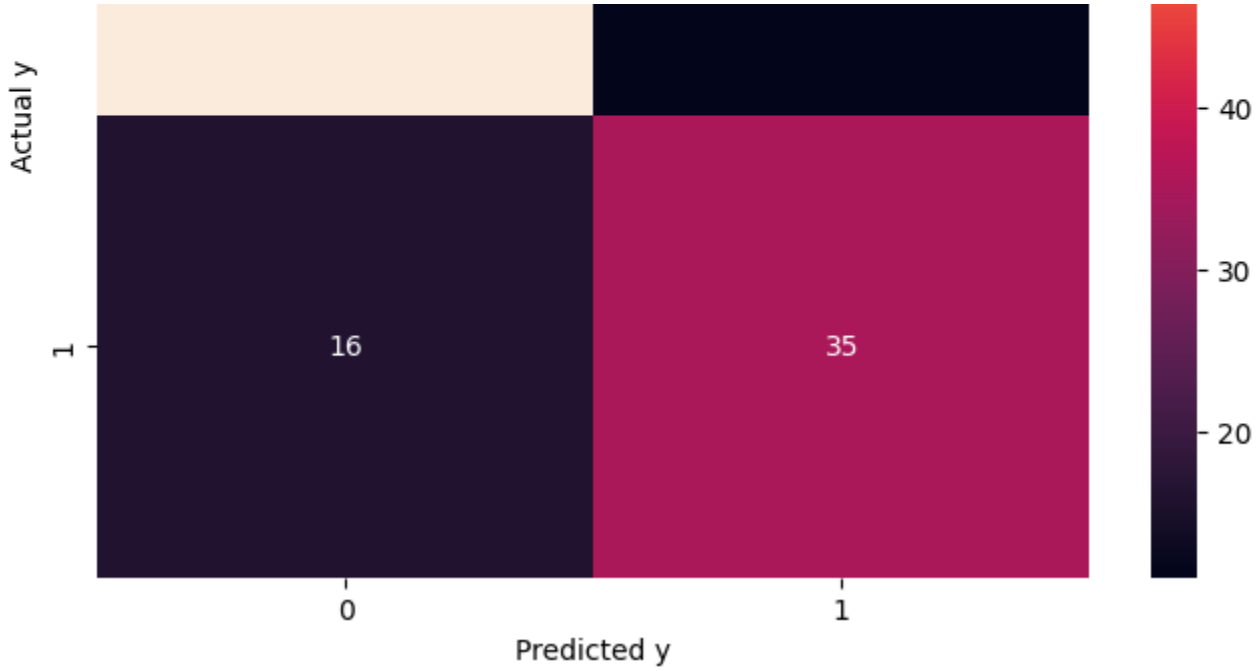
```
n_estimators=1, avg accuracy: 0.6799891067538126
n_estimators=1, train accuracy: 0.8864306784660767
n_estimators=1, test accuracy: 0.6461538461538462
n_estimators=10, avg accuracy: 0.7625381263616557
n_estimators=10, train accuracy: 0.9690265486725663
n_estimators=10, test accuracy: 0.7692307692307693
n_estimators=15, avg accuracy: 0.7625272331154684
n_estimators=15, train accuracy: 0.9734513274336283
```

```
n_estimators=15, test accuracy: 0.7692307692307693
n_estimators=25, avg accuracy: 0.7654901960784313
n_estimators=25, train accuracy: 0.9793510324483776
n_estimators=25, test accuracy: 0.7846153846153846
n_estimators=50, avg accuracy: 0.7743355119825709
n_estimators=50, train accuracy: 0.9793510324483776
n_estimators=50, test accuracy: 0.7846153846153846
n_estimators=100, avg accuracy: 0.7802178649237473
n_estimators=100, train accuracy: 0.9808259587020649
n_estimators=100, test accuracy: 0.7923076923076923
```




```
print(f"Best Random Forest accuracy: {accuracy_score(y_test, y_pred)}")
create_confusion_matrix(y_test, y_pred, "Random Forest")
```

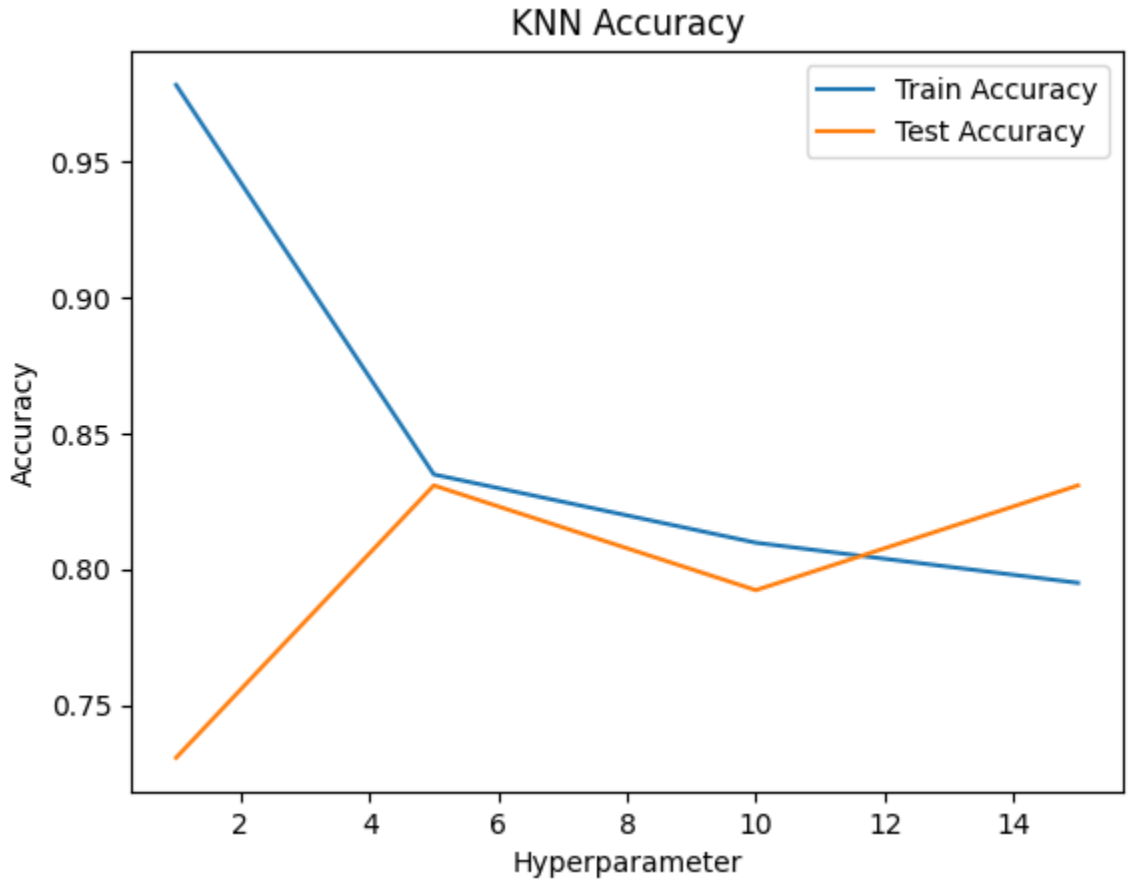
```
Best Random Forest accuracy: 0.7923076923076923
```




```
def knn_model(train_pd: pd.DataFrame, test_pd: pd.DataFrame) -> tuple:
    X_train = train_pd["features"].tolist()
    X_test = test_pd["features"].tolist()
    y_train = train_pd["Label"].tolist()
    y_test = test_pd["Label"].tolist()
    test_accuracy = []
    train_accuracy = []
    best_model = None
    best_score = 0

    for n in [1,5,10,15]:
        model = KNeighborsClassifier(n_neighbors=n)
        scores = cross_val_score(model, X_train, y_train, cv=5)
        print(f"n_neighbors={n}, avg accuracy: {scores.mean()}")
        model.fit(X_train, y_train)

        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)

        train_acc = accuracy_score(y_train, train_pred)
        test_acc = accuracy_score(y_test, test_pred)
        print(f"n_neighbors={n}, train accuracy: {train_acc}")
        print(f"n_neighbors={n}, test accuracy: {test_acc}")
        train_accuracy.append(train_acc)
        test_accuracy.append(test_acc)

        if test_acc > best_score:
            best_score = test_acc
            best_model = model
    best_pred = best_model.predict(X_test)
```

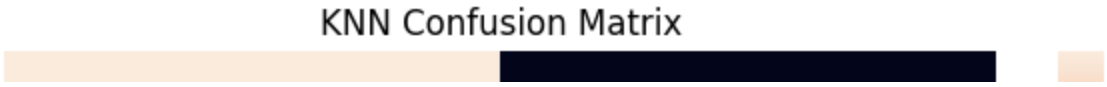
```
    best_pred = best_model.predict(x_test)

    return best_pred, train_accuracy, test_accuracy, y_test

y_pred, train_accuracy, test_accuracy, y_test = knn_model(train_pd, test_pd)
plot_accuracy([1,5,10,15], train_accuracy, test_accuracy, "KNN")
```

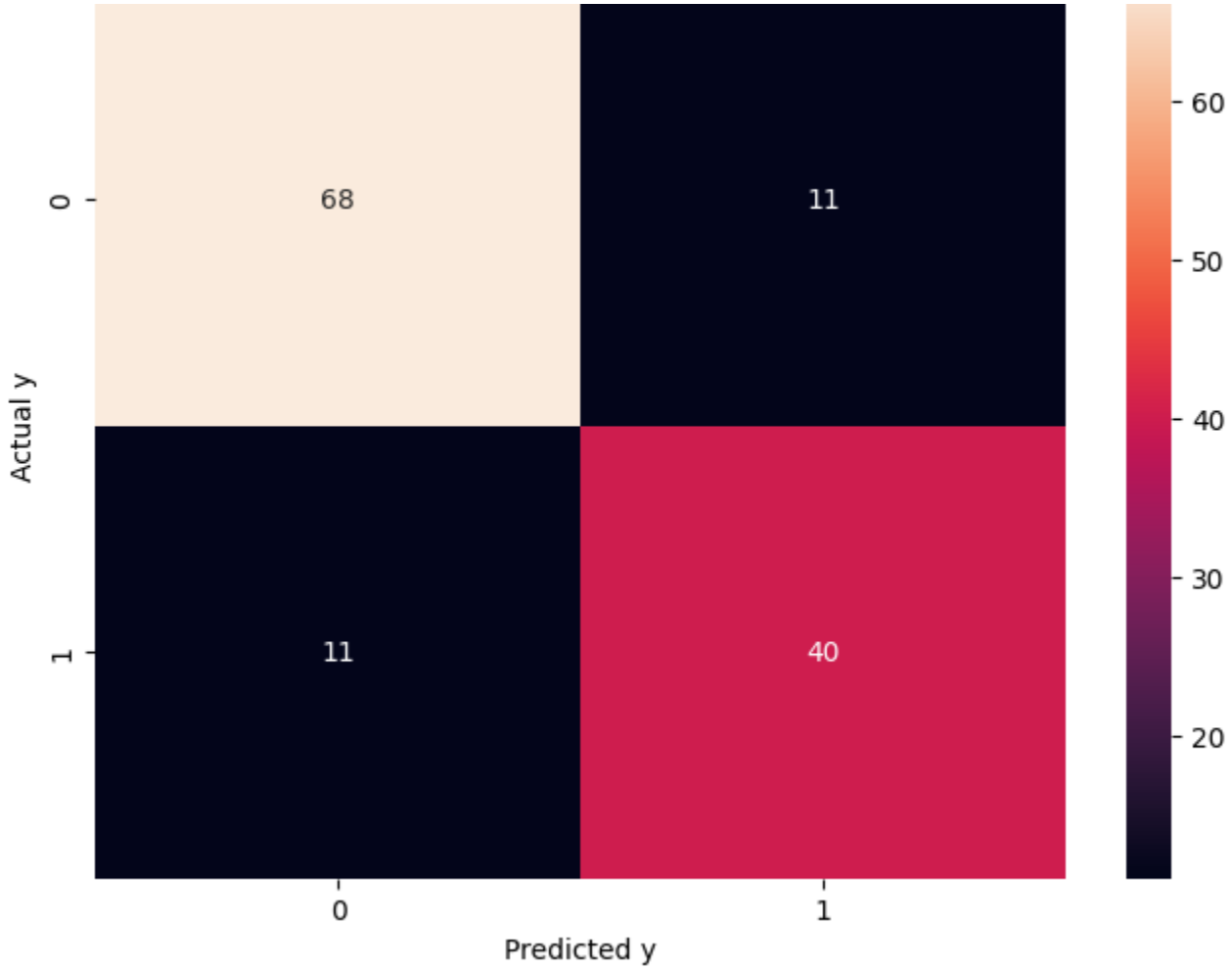
```
    n_neighbors=1, avg accuracy: 0.7226797385620916
    n_neighbors=1, train accuracy: 0.9778761061946902
    n_neighbors=1, test accuracy: 0.7307692307692307
    n_neighbors=5, avg accuracy: 0.7581263616557734
    n_neighbors=5, train accuracy: 0.8348082595870207
    n_neighbors=5, test accuracy: 0.8307692307692308
    n_neighbors=10, avg accuracy: 0.7655119825708061
    n_neighbors=10, train accuracy: 0.8097345132743363
    n_neighbors=10, test accuracy: 0.7923076923076923
    n_neighbors=15, avg accuracy: 0.758169934640523
    n_neighbors=15, train accuracy: 0.7949852507374632
    n_neighbors=15, test accuracy: 0.8307692307692308
```




```
print(f"Best KNN accuracy: {accuracy_score(y_test, y_pred)}")
create_confusion_matrix(y_test, y_pred, "KNN")
```

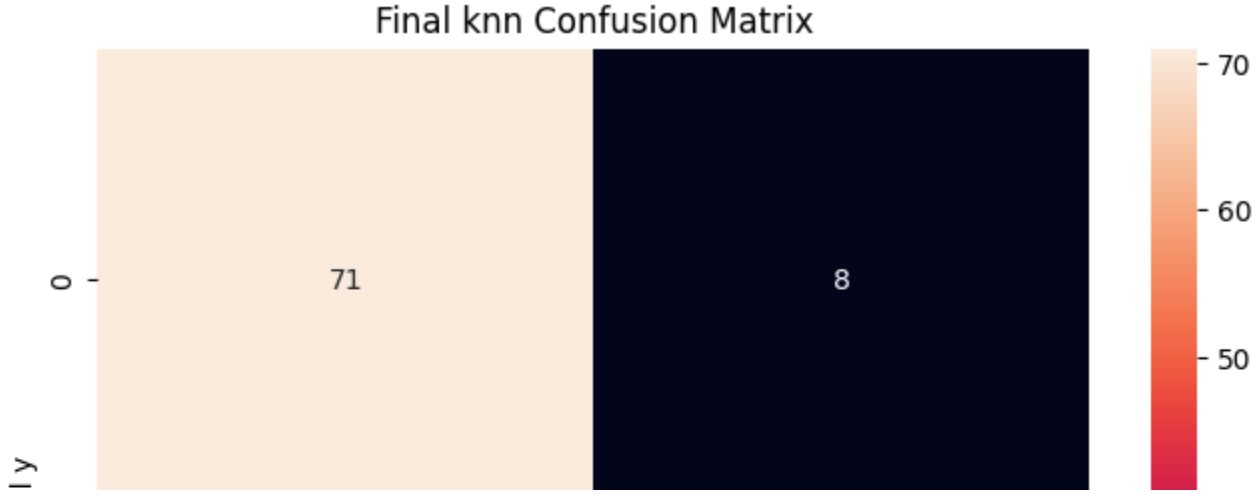
```
    Best KNN accuracy: 0.8307692307692308
```

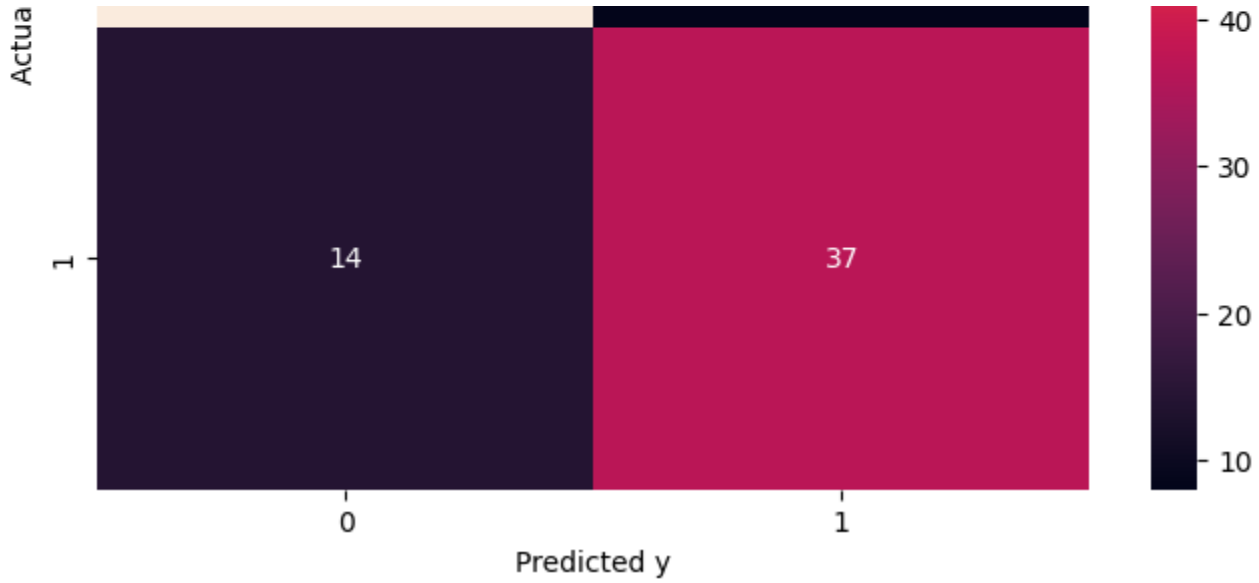





```
final_model = KNeighborsClassifier(n_neighbors=15)
final_model.fit(train_pd["features"].tolist(), train_pd["Label"].tolist())
y_pred = final_model.predict(test_pd["features"].tolist())
accuracy = accuracy_score(test_pd["Label"].tolist(), y_pred)
print(f"Final accuracy: {accuracy}")
create_confusion_matrix(test_pd["Label"].tolist(), y_pred, "Final knn")
```

```
Final accuracy: 0.8307692307692308
```





Write your report here

After loading the data, I conducted initial data analysis to understand the dataset better. The analysis revealed that the dataset had 891 samples and 8 features excluding the PassengerId, Survived, Name, and Ticket columns. I decided to drop Name and Ticket as I think that they don't have any impact on survival. The cabin column had a lot missing values and there were some unreasonable values in the age, parents/children aboard, siblings/spouses aboard, and far columns such as negative ages, negative number of sibling/spouses aboard, fare value of 0.

To clean the data, I first replaced missing age values with the average age of passengers excluding the unreasonable values. For the cabin column, I noticed that first class passengers paid more than 35.5 and had cabin numbers that started with A up to D,while second class passengers paid between 10.5 and 35.4 and had cabin numbers that started with D up to F and third class passengers paid less than 10.5 and had cabin numbers that started with F up to G. I used this information to fill the missing cabin values and missing passenger class values. Giving first class passengers cabin number 'A', second class passengers cabin number 'B', and third class passengers cabin number 'C'. For all other features, I replaced missing values with the mode of the respective feature. I removed three duplicate samples from the dataset. I removed outliers using the 3 standard deviation method for numerical features, setting specific constraints like limiting ages to 0-100 years and fares to less than 350.

After cleaning the data, I split the dataset into a training set and a test set, using 80-20 split and then standardized numeric features using the StandardScaler to imporve permanence. I encoded the categorical features using the OneHotEncoder. I then trained the models using the training set and tested the models using the test set.

I implemented and compared four models; SVM, Logistic Regression, Random Forest, and K-Nearest Neighbors. Each model underwent Cross Validation with varying hyperparameters to find the optimal parameter. SVM model had an accuracy of 80.77%, Logistic Regression had an accuracy of 80%, Random Forest had an accuracy of 79.23%, and K-Nearest Neighbors had an accuracy of 83.08%. I selected K-Nearest Neighbors as the final model since it had the highest accuracy.

Double-click (or enter) to edit
