В приведенном примере решалась задача классификации. Цель заключалась в предсказании выживаемости пассажиров Титаника (переменная Survived), которая принимает значения 0 (не выжил) или 1 (выжил).

Это классическая задача бинарной классификации, так как у нас есть два возможных класса.

Для уточнения, вот ключевые аспекты, указывающие на то, что это задача классификации:

Целевая переменная (Survived): бинарная переменная, принимающая значения 0 или 1. Метрика оценки: использовалась метрика точности (accuracy), которая подходит для задач классификации.

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder
    from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
    from sklearn.metrics import accuracy_score
    from sklearn.dummy import DummyClassifier
    # Шаг 1: Загрузка данных
    url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
    df = pd.read_csv(url)

    # Просмотр первых строк данных
    print(df.head())
```

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

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

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
# Шаг 2: Предобработка данных
# Заполнение пропусков
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df['Fare'].fillna(df['Fare'].median(), inplace=True)

# Удаление ненужных столбцов
df.drop(columns=['Cabin', 'Name', 'Ticket', 'PassengerId'], inplace=True)

# Кодирование категориальных признаков
label_enc = LabelEncoder()
df['Sex'] = label_enc.fit_transform(df['Sex'])
df['Embarked'] = label_enc.fit_transform(df['Embarked'])

# Разделение данных на признаки и метки
X = df.drop('Survived', axis=1)
y = df['Survived']
print(df.head())
```

```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0         0       3    1  22.0      1      0   7.2500         2
1         1       1    0  38.0      1      0  71.2833         0
2         1       3    0  26.0      0      0   7.9250         2
3         1       1    0  35.0      1      0  53.1000         2
4         0       3    1  35.0      0      0   8.0500         2
```

```
# Шаг 3: Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Шаг 4: Обучение моделей
# Модель бэггинга
bagging_model = BaggingClassifier(n_estimators=100, random_state=42)
bagging_model.fit(X_train, y_train)

# Модель случайного леса
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Модель AdaBoost
ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_model.fit(X_train, y_train)

# Модель градиентного бустинга
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train, y_train)

# Добавление DummyClassifier для базовой линии
dummy_model = DummyClassifier(strategy="most_frequent")
dummy_model.fit(X_train, y_train)
```

```
DummyClassifier(strategy='most_frequent')
```

```
# Шаг 5: Оценка моделей
# Предсказания на тестовой выборке
y_pred_bagging = bagging_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
y_pred_ada = ada_model.predict(X_test)
y_pred_gb = gb_model.predict(X_test)
y_pred_dummy = dummy_model.predict(X_test)

# Оценка точности моделей
accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_ada = accuracy_score(y_test, y_pred_ada)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
accuracy_dummy = accuracy_score(y_test, y_pred_dummy)

# Вывод результатов
print(f'Bagging accuracy: {accuracy_bagging:.4f}')
print(f'Random Forest accuracy: {accuracy_rf:.4f}')
print(f'AdaBoost accuracy: {accuracy_ada:.4f}')
print(f'Gradient Boosting accuracy: {accuracy_gb:.4f}')
print(f'Dummy accuracy: {accuracy_dummy:.4f}')
```

```
Bagging accuracy: 0.8101
Random Forest accuracy: 0.8212
AdaBoost accuracy: 0.8101
Gradient Boosting accuracy: 0.8101
Dummy accuracy: 0.5866
```

DummyClassifier (accuracy_dummy): Этот классификатор служит базовой линией, предсказывая самый частый класс. Его точность показывает, как часто модель правильно угадывает исходя из частоты классов без учета признаков.

BaggingClassifier (accuracy_bagging): Модель бэггинга, объединяющая несколько слабых моделей для улучшения устойчивости и точности.

RandomForestClassifier (accuracy_rf): Модель случайного леса, использующая ансамбль решений деревьев для повышения точности и уменьшения переобучения.

AdaBoostClassifier (accuracy_ada): Модель AdaBoost, которая последовательно улучшает ошибки предыдущих классификаторов.

GradientBoostingClassifier (accuracy_gb): Модель градиентного бустинга, обучающая каждое последующее дерево на ошибках предыдущих деревьев для улучшения предсказаний.