Machine Learing HW

linear regression as a classifier

20213064_김종민

In [237]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [238]:
```
train_data = pd.read_csv("train.csv") # train_data 읽어오기
test_data = pd.read_csv("test.csv") # test_data 읽어오기
train_data.info(), test_data.info() # 각각의 data들의 information 확인
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   891 non-null    int64
 1   PassengerId  891 non-null    int64
 2   Survived     891 non-null    int64
 3   Pclass       891 non-null    int64
 4   Name         891 non-null    object
 5   Sex          891 non-null    object
 6   Age          714 non-null    float64
 7   SibSp        891 non-null    int64
 8   Parch        891 non-null    int64
 9   Ticket       891 non-null    object
 10  Fare         891 non-null    float64
 11  Cabin        204 non-null    object
 12  Embarked     889 non-null    object
dtypes: float64(2), int64(6), object(5)
memory usage: 90.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   418 non-null    int64
 1   PassengerId  418 non-null    int64
 2   Survived     418 non-null    int64
 3   Pclass       418 non-null    int64
 4   Name         418 non-null    object
 5   Sex          418 non-null    object
 6   Age          332 non-null    float64
 7   SibSp        418 non-null    int64
 8   Parch        418 non-null    int64
 9   Ticket       418 non-null    object
 10  Fare         417 non-null    float64
 11  Cabin        91 non-null     object
 12  Embarked     418 non-null    object
dtypes: float64(2), int64(6), object(5)
memory usage: 42.6+ KB
```

Out[238]:
```
(None, None)
```

In [239]:
```
# 결측치 확인하기
missingTrain, missingTest = train_data.isnull().sum(), test_data.isnull().sum()
missing_data = pd.DataFrame({
    "Train Missing": missingTrain,
    "Test Missing": missingTest
})
missing_data
```

Out[239]:

	Train Missing	Test Missing
Unnamed: 0	0	0
PassengerId	0	0
Survived	0	0
Pclass	0	0
Name	0	0
Sex	0	0
Age	177	86
SibSp	0	0
Parch	0	0
Ticket	0	0
Fare	0	1
Cabin	687	327
Embarked	2	0

In [240]:
```
for df in [train_data, test_data]:
    df['Sex'] = df['Sex'].map({'male': 0, 'female' : 1}) # 레이블 인코딩
    df['Age'].fillna(df['Age'].mean(), inplace = True) # 나이 결측치를 평균으로 대체
    df['Fare'].fillna(df['Fare'].mean(), inplace = True) # 임금 결측치를 평균으로 대체
    df['Embarked'].fillna(df['Embarked'].mode()[0], inplace = True) # 승선한 곳을 최빈값으로 대체
    df['Cabin'].fillna('Unknown', inplace = True) # 사실 필요없는 정보라서 Unknown으로 대체
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare'] # 6개의 features를 사용할 것임
X = train_data[features]
y = train_data['Survived']
Xt = test_data[features]
yt = test_data['Survived']
```

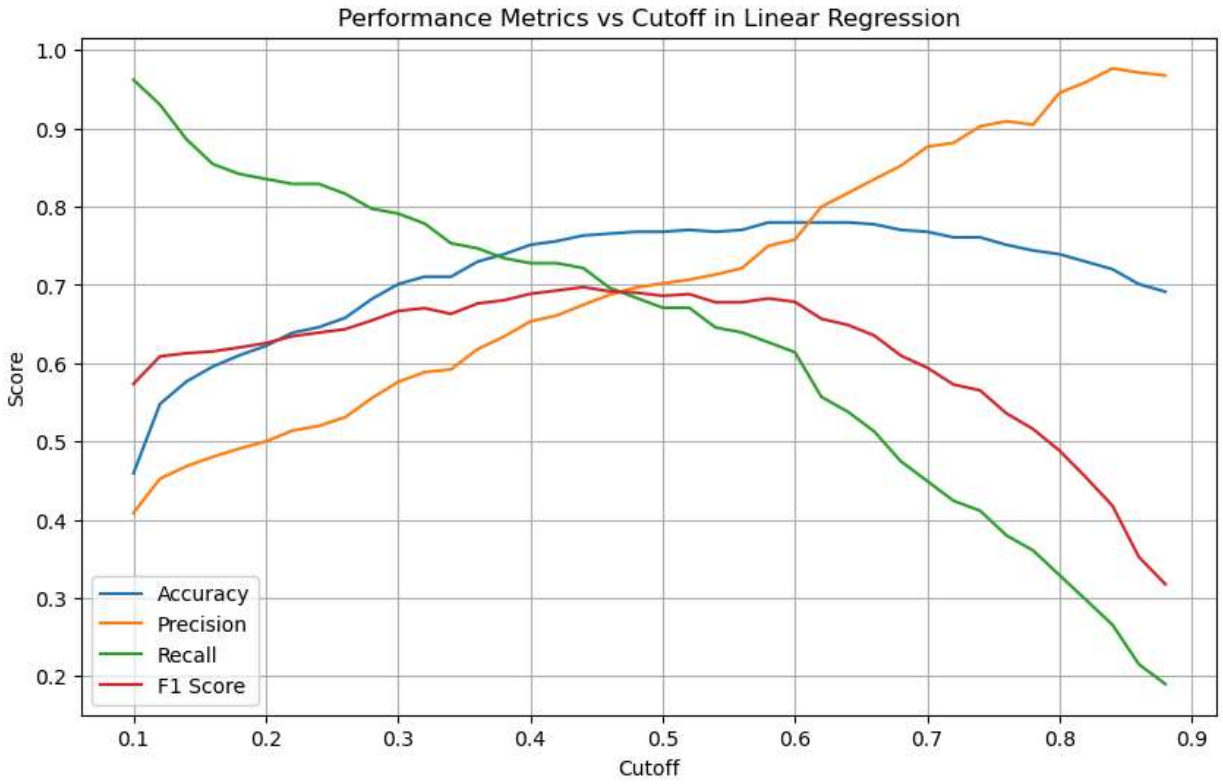
In [241]:
```
# 선형 회귀 모델 학습
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(Xt)
```

In [248]:
```
import numpy as np
cutoffs = np.arange(0.1, 0.9, 0.02)
best_f1 = 0
best_cutoff = 0
accuracies = []
precisions = []
recalls = []
f1_scores = []
for cutoff in cutoffs:
    y_pred_class = [1 if y >= cutoff else 0 for y in y_pred]
    # accuracy, precision, recall, f1 score 계산
    acc = accuracy_score(yt, y_pred_class)
    precision = precision_score(yt, y_pred_class)
    recall = recall_score(yt, y_pred_class)
    f1 = f1_score(yt, y_pred_class)

    accuracies.append(acc)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    print(f"Cutoff: {cutoff:.2f}, Accuracy: {acc:.4f}, Precision: {precision:.4f}, W
Recall: {recall:.4f}, F1 Score: {f1:.4f}")

    if f1 > best_f1:
        best_f1 = f1
        best_cutoff = cutoff

print(f"₩n최적의 Cutoff: {best_cutoff}, F1 Score: {best_f1}")
```

```
Cutoff: 0.10, Accuracy: 0.4593, Precision: 0.4086, Recall: 0.9620, F1 Score: 0.5736
Cutoff: 0.12, Accuracy: 0.5478, Precision: 0.4523, Recall: 0.9304, F1 Score: 0.6087
Cutoff: 0.14, Accuracy: 0.5766, Precision: 0.4682, Recall: 0.8861, F1 Score: 0.6127
Cutoff: 0.16, Accuracy: 0.5957, Precision: 0.4804, Recall: 0.8544, F1 Score: 0.6150
Cutoff: 0.18, Accuracy: 0.6100, Precision: 0.4908, Recall: 0.8418, F1 Score: 0.6200
Cutoff: 0.20, Accuracy: 0.6220, Precision: 0.5000, Recall: 0.8354, F1 Score: 0.6256
Cutoff: 0.22, Accuracy: 0.6388, Precision: 0.5137, Recall: 0.8291, F1 Score: 0.6344
Cutoff: 0.24, Accuracy: 0.6459, Precision: 0.5198, Recall: 0.8291, F1 Score: 0.6390
Cutoff: 0.26, Accuracy: 0.6579, Precision: 0.5309, Recall: 0.8165, F1 Score: 0.6434
Cutoff: 0.28, Accuracy: 0.6818, Precision: 0.5551, Recall: 0.7975, F1 Score: 0.6545
Cutoff: 0.30, Accuracy: 0.7010, Precision: 0.5760, Recall: 0.7911, F1 Score: 0.6667
Cutoff: 0.32, Accuracy: 0.7105, Precision: 0.5885, Recall: 0.7785, F1 Score: 0.6703
Cutoff: 0.34, Accuracy: 0.7105, Precision: 0.5920, Recall: 0.7532, F1 Score: 0.6630
Cutoff: 0.36, Accuracy: 0.7297, Precision: 0.6178, Recall: 0.7468, F1 Score: 0.6762
Cutoff: 0.38, Accuracy: 0.7392, Precision: 0.6339, Recall: 0.7342, F1 Score: 0.6804
Cutoff: 0.40, Accuracy: 0.7512, Precision: 0.6534, Recall: 0.7278, F1 Score: 0.6886
Cutoff: 0.42, Accuracy: 0.7560, Precision: 0.6609, Recall: 0.7278, F1 Score: 0.6928
Cutoff: 0.44, Accuracy: 0.7632, Precision: 0.6746, Recall: 0.7215, F1 Score: 0.6972
Cutoff: 0.46, Accuracy: 0.7656, Precision: 0.6875, Recall: 0.6962, F1 Score: 0.6918
Cutoff: 0.48, Accuracy: 0.7679, Precision: 0.6968, Recall: 0.6835, F1 Score: 0.6901
Cutoff: 0.50, Accuracy: 0.7679, Precision: 0.7020, Recall: 0.6709, F1 Score: 0.6861
Cutoff: 0.52, Accuracy: 0.7703, Precision: 0.7067, Recall: 0.6709, F1 Score: 0.6883
Cutoff: 0.54, Accuracy: 0.7679, Precision: 0.7133, Recall: 0.6456, F1 Score: 0.6777
Cutoff: 0.56, Accuracy: 0.7703, Precision: 0.7214, Recall: 0.6392, F1 Score: 0.6779
Cutoff: 0.58, Accuracy: 0.7799, Precision: 0.7500, Recall: 0.6266, F1 Score: 0.6828
Cutoff: 0.60, Accuracy: 0.7799, Precision: 0.7578, Recall: 0.6139, F1 Score: 0.6783
Cutoff: 0.62, Accuracy: 0.7799, Precision: 0.8000, Recall: 0.5570, F1 Score: 0.6567
Cutoff: 0.64, Accuracy: 0.7799, Precision: 0.8173, Recall: 0.5380, F1 Score: 0.6489
Cutoff: 0.66, Accuracy: 0.7775, Precision: 0.8351, Recall: 0.5127, F1 Score: 0.6353
Cutoff: 0.68, Accuracy: 0.7703, Precision: 0.8523, Recall: 0.4747, F1 Score: 0.6098
Cutoff: 0.70, Accuracy: 0.7679, Precision: 0.8765, Recall: 0.4494, F1 Score: 0.5941
Cutoff: 0.72, Accuracy: 0.7608, Precision: 0.8816, Recall: 0.4241, F1 Score: 0.5726
Cutoff: 0.74, Accuracy: 0.7608, Precision: 0.9028, Recall: 0.4114, F1 Score: 0.5652
Cutoff: 0.76, Accuracy: 0.7512, Precision: 0.9091, Recall: 0.3797, F1 Score: 0.5357
Cutoff: 0.78, Accuracy: 0.7440, Precision: 0.9048, Recall: 0.3608, F1 Score: 0.5158
Cutoff: 0.80, Accuracy: 0.7392, Precision: 0.9455, Recall: 0.3291, F1 Score: 0.4883
Cutoff: 0.82, Accuracy: 0.7297, Precision: 0.9592, Recall: 0.2975, F1 Score: 0.4541
Cutoff: 0.84, Accuracy: 0.7201, Precision: 0.9767, Recall: 0.2658, F1 Score: 0.4179
Cutoff: 0.86, Accuracy: 0.7010, Precision: 0.9714, Recall: 0.2152, F1 Score: 0.3523
Cutoff: 0.88, Accuracy: 0.6914, Precision: 0.9677, Recall: 0.1899, F1 Score: 0.3175

최적의 Cutoff: 0.44000000000000006, F1 Score: 0.6972477064220183
```

In [249]:
```
y_pred_optimal = [1 if y >= best_cutoff else 0 for y in y_pred]

final_acc = accuracy_score(yt, y_pred_optimal)
final_precision = precision_score(yt, y_pred_optimal)
final_recall = recall_score(yt, y_pred_optimal)
final_f1 = f1_score(yt, y_pred_optimal)

print("최적 모델 성능")
print(f"Accuracy: {final_acc:.4f}, Precision: {final_precision:.4f}, W
Recall: {final_recall:.4f}, F1 Score: {final_f1:.4f}")
plt.figure(figsize=(10, 6))
plt.plot(cutoffs, accuracies, label='Accuracy')
plt.plot(cutoffs, precisions, label='Precision')
plt.plot(cutoffs, recalls, label='Recall')
plt.plot(cutoffs, f1_scores, label='F1 Score')
plt.xlabel('Cutoff')
plt.ylabel('Score')
plt.title('Performance Metrics vs Cutoff in Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
최적 모델 성능
Accuracy: 0.7632, Precision: 0.6746, Recall: 0.7215, F1 Score: 0.6972
```



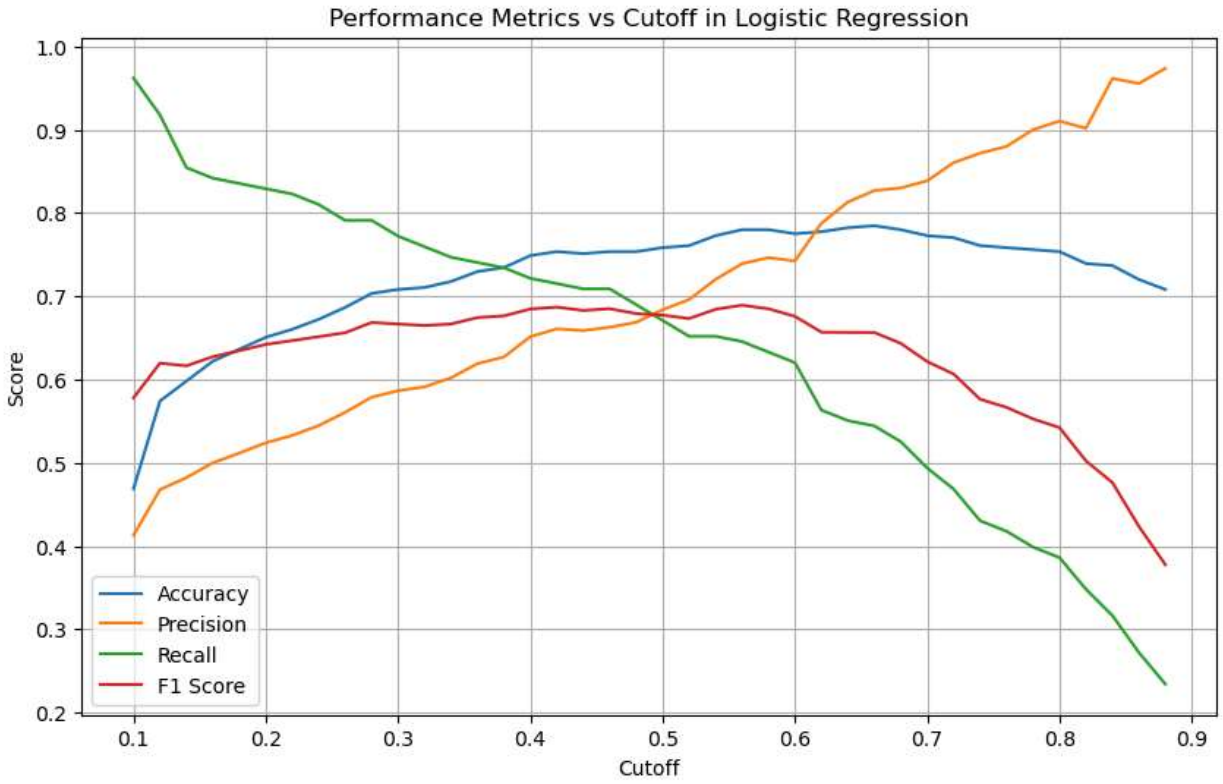
In [244]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# 로지스틱 회귀 모델 학습
model_logreg = LogisticRegression()
model_logreg.fit(X, y)
y_pred_logreg = model_logreg.predict_proba(Xt)[:, 1]
```

In [250]:
```
cutoffs = np.arange(0.1, 0.9, 0.02)
best_f1 = 0
best_cutoff = 0
accuracies = []
precisions = []
recalls = []
f1_scores = []
for cutoff in cutoffs:
    y_pred_class_logreg = [1 if y >= cutoff else 0 for y in y_pred_logreg]
    # accuracy, precision, recall, f1 score 계산
    acc = accuracy_score(yt, y_pred_class_logreg)
    precision = precision_score(yt, y_pred_class_logreg)
    recall = recall_score(yt, y_pred_class_logreg)
    f1 = f1_score(yt, y_pred_class_logreg)

    accuracies.append(acc)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)
    print(f"Cutoff: {cutoff:.2f}, Accuracy: {acc:.4f}, Precision: {precision:.4f}, W
Recall: {recall:.4f}, F1 Score: {f1:.4f}")

    if f1 > best_f1:
        best_f1 = f1
        best_cutoff = cutoff

print(f"Wn최적의 Cutoff: {best_cutoff}, F1 Score: {best_f1}")
```

```
Cutoff: 0.10, Accuracy: 0.4689, Precision: 0.4130, Recall: 0.9620, F1 Score: 0.5779
Cutoff: 0.12, Accuracy: 0.5742, Precision: 0.4677, Recall: 0.9177, F1 Score: 0.6197
Cutoff: 0.14, Accuracy: 0.5981, Precision: 0.4821, Recall: 0.8544, F1 Score: 0.6164
Cutoff: 0.16, Accuracy: 0.6220, Precision: 0.5000, Recall: 0.8418, F1 Score: 0.6274
Cutoff: 0.18, Accuracy: 0.6364, Precision: 0.5116, Recall: 0.8354, F1 Score: 0.6346
Cutoff: 0.20, Accuracy: 0.6507, Precision: 0.5240, Recall: 0.8291, F1 Score: 0.6422
Cutoff: 0.22, Accuracy: 0.6603, Precision: 0.5328, Recall: 0.8228, F1 Score: 0.6468
Cutoff: 0.24, Accuracy: 0.6722, Precision: 0.5447, Recall: 0.8101, F1 Score: 0.6514
Cutoff: 0.26, Accuracy: 0.6866, Precision: 0.5605, Recall: 0.7911, F1 Score: 0.6562
Cutoff: 0.28, Accuracy: 0.7033, Precision: 0.5787, Recall: 0.7911, F1 Score: 0.6684
Cutoff: 0.30, Accuracy: 0.7081, Precision: 0.5865, Recall: 0.7722, F1 Score: 0.6667
Cutoff: 0.32, Accuracy: 0.7105, Precision: 0.5911, Recall: 0.7595, F1 Score: 0.6648
Cutoff: 0.34, Accuracy: 0.7177, Precision: 0.6020, Recall: 0.7468, F1 Score: 0.6667
Cutoff: 0.36, Accuracy: 0.7297, Precision: 0.6190, Recall: 0.7405, F1 Score: 0.6744
Cutoff: 0.38, Accuracy: 0.7344, Precision: 0.6270, Recall: 0.7342, F1 Score: 0.6764
Cutoff: 0.40, Accuracy: 0.7488, Precision: 0.6514, Recall: 0.7215, F1 Score: 0.6847
Cutoff: 0.42, Accuracy: 0.7536, Precision: 0.6608, Recall: 0.7152, F1 Score: 0.6869
Cutoff: 0.44, Accuracy: 0.7512, Precision: 0.6588, Recall: 0.7089, F1 Score: 0.6829
Cutoff: 0.46, Accuracy: 0.7536, Precision: 0.6627, Recall: 0.7089, F1 Score: 0.6850
Cutoff: 0.48, Accuracy: 0.7536, Precision: 0.6687, Recall: 0.6899, F1 Score: 0.6791
Cutoff: 0.50, Accuracy: 0.7584, Precision: 0.6839, Recall: 0.6709, F1 Score: 0.6773
Cutoff: 0.52, Accuracy: 0.7608, Precision: 0.6959, Recall: 0.6519, F1 Score: 0.6732
Cutoff: 0.54, Accuracy: 0.7727, Precision: 0.7203, Recall: 0.6519, F1 Score: 0.6844
Cutoff: 0.56, Accuracy: 0.7799, Precision: 0.7391, Recall: 0.6456, F1 Score: 0.6892
Cutoff: 0.58, Accuracy: 0.7799, Precision: 0.7463, Recall: 0.6329, F1 Score: 0.6849
Cutoff: 0.60, Accuracy: 0.7751, Precision: 0.7424, Recall: 0.6203, F1 Score: 0.6759
Cutoff: 0.62, Accuracy: 0.7775, Precision: 0.7876, Recall: 0.5633, F1 Score: 0.6568
Cutoff: 0.64, Accuracy: 0.7823, Precision: 0.8131, Recall: 0.5506, F1 Score: 0.6566
Cutoff: 0.66, Accuracy: 0.7847, Precision: 0.8269, Recall: 0.5443, F1 Score: 0.6565
Cutoff: 0.68, Accuracy: 0.7799, Precision: 0.8300, Recall: 0.5253, F1 Score: 0.6434
Cutoff: 0.70, Accuracy: 0.7727, Precision: 0.8387, Recall: 0.4937, F1 Score: 0.6215
Cutoff: 0.72, Accuracy: 0.7703, Precision: 0.8605, Recall: 0.4684, F1 Score: 0.6066
Cutoff: 0.74, Accuracy: 0.7608, Precision: 0.8718, Recall: 0.4304, F1 Score: 0.5763
Cutoff: 0.76, Accuracy: 0.7584, Precision: 0.8800, Recall: 0.4177, F1 Score: 0.5665
Cutoff: 0.78, Accuracy: 0.7560, Precision: 0.9000, Recall: 0.3987, F1 Score: 0.5526
Cutoff: 0.80, Accuracy: 0.7536, Precision: 0.9104, Recall: 0.3861, F1 Score: 0.5422
Cutoff: 0.82, Accuracy: 0.7392, Precision: 0.9016, Recall: 0.3481, F1 Score: 0.5023
Cutoff: 0.84, Accuracy: 0.7368, Precision: 0.9615, Recall: 0.3165, F1 Score: 0.4762
Cutoff: 0.86, Accuracy: 0.7201, Precision: 0.9556, Recall: 0.2722, F1 Score: 0.4236
Cutoff: 0.88, Accuracy: 0.7081, Precision: 0.9737, Recall: 0.2342, F1 Score: 0.3776

최적의 Cutoff: 0.56, F1 Score: 0.6891891891891891
```

In [246]:
```
y_pred_optimal_logr = [1 if y >= best_cutoff else 0 for y in y_pred_logreg]

final_acc = accuracy_score(yt, y_pred_optimal_logr)
final_precision = precision_score(yt, y_pred_optimal_logr)
final_recall = recall_score(yt, y_pred_optimal_logr)
final_f1 = f1_score(yt, y_pred_optimal_logr)

print("최적 모델 성능")
print(f"Accuracy: {final_acc:.4f}, Precision: {final_precision:.4f},W
Recall: {final_recall:.4f}, F1 Score: {final_f1:.4f}")
plt.figure(figsize=(10, 6))
plt.plot(cutoffs, accuracies, label='Accuracy')
plt.plot(cutoffs, precisions, label='Precision')
plt.plot(cutoffs, recalls, label='Recall')
plt.plot(cutoffs, f1_scores, label='F1 Score')
plt.xlabel('Cutoff')
plt.ylabel('Score')
plt.title('Performance Metrics vs Cutoff in Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
최적 모델 성능
Accuracy: 0.7799, Precision: 0.7391,Recall: 0.6456, F1 Score: 0.6892
```



결과 분석

위의 결과들을 토대로 선형 회귀와 로지스틱 회귀 중 어떤 것의 성능이 더 나은 지와 왜 성능 차이가 나는지에 대해서 말씀 드리겠습니다.

우선 Linear Regrssion **모델을 사용한 결과** accuracy**가** 0.7632, precision**이** 0.6746, recall**이** 0.7215, F1 Score**가** 0.6972**였습니다. 한편** Logistic Regression **모델을 사용한 결과** accuracy**는** 0.7799, precision**이** 0.7391, recall**이** 0.6456, F1 Score**가** 0.6892**였습니다.**

수치에서도 확인할 수 있듯이 로지스틱 회귀에서는 accuracy**와** precision**이 더 높은 반면에** recall**과** F1 Score**는 선형 회귀가 더 높은 것을 알 수 있습니다.**

성능 차이가 나는 이유에 대해 말씀드리겠습니다. 선형회귀는 모델이 예측값을 연속적으로 출력하기 때문에, 입력 값의 변화에 따라 출력이 크게 변할 수 있습니다.예를 들어 선형 회귀의 경우 입력 특성들이 변화하면 모델의 예측 값도 동일한 변화 비율도 크게 바뀔 수 있습니다. 이 때문에 선형 회귀 모델이 이진 분류에 사용될 때, 경계 값을 임의로 설정해서 분류를 수행하더라도 출력 값이 넓은 범위에 걸쳐 분포될 수 있습니다. 결과적으로, 로지스틱 회귀처럼 분류 경계가 명확하지 않더라도 특정 상황에서는 유사한 성능을 보일 수 있습니다.

반면 로지스틱 회귀는 시그모이드 함수를 사용해 0과 1 사이의 확률로 예측 값을 제한하며, 확률로 해석 가능한 출력을 제공합니다. 즉, 로지스틱 회귀는 분류 문제에 특화된 방식으로 예측값이 0과 1사이에서 안정적인 형태로 변동합니다. 선형 회귀와 로지스틱 회귀의 성능 차이는 다시 말해서 입력 데이터 변화에 따라 모델이 출력하는 값이 얼마나 크게 변동하는지를 의미하는 "민감도"에서 기인합니다.

선형 회귀와 로지스틱 회귀 중에 현재의 문제에 대하여 어떤 것의 성능이 더 좋은지를 판단하라 할 때, 현수치에서는 어느 쪽이 명확히 좋다라고 말씀드릴 수 없지만 굳이 말하자면, 로지스틱 회귀가 성능이 더 좋다고 할 수 있습니다, 왜냐하면 근본적으로 문제의 특성상 이진 분류이기 때문에 로지스틱 회귀가 더 적합하기도 하고, recall은 낮을지라도 이와 같은 이진 분류에서는 accuracy와 precision이 더 중요한 지표라고 생각되기 때문입니다.

In [ ]: