통계적 학습의 원리와 이해 I 시험 (2025)

1. 코딩/분석을 필요로 하는 문제의 경우 실행후 나온 결과값을 기반으로 Markdown Cell 에 결과값 해석 및 설명
2. 일반 문제의 경우 답안용 Markdown cell 에 그대로 편집해서 작성. (답) 이라고 된 부분.

학번/이름 작성

- **학번**: 2025729110
- **이름**: 박철준

Q1. (10점) True/False Question: True/False와 그 이유도 함께 작성

1. 복잡한 비선형 방법이 언제나 단순한 모형보다 좋다.

(답): **False**. 단순한 모형이 비선형 모델보다 예측력은 떨어질 수도 있으나, 설명력이 더 좋고 Overfitting의 위험도가 낮다.

2. 일반적으로 less flexible 한 모델이 variance 가 크다.

(답): **False**. 유연성이 적은 모델은 bias가 크고, variance가 작다. 대표적인 케이스가 선형회귀모델이다.

3. Training MSE 는 매우 flexible 한 모델의 경우 0이 되기도 한다.

(답): **True**. 모델이 매우 유연하면 모든 학습 데이터를 완벽하게 맞출 수 있어 Training MSE가 0이 될 수 있다. 그러나 이 경우 Overfitting 되었을 가능성이 매우 높다.

4. 좋은 Classifier 라는 것은 Training Error가 가장 낮은 것을 의미한다.

(답): **False**. Training Error가 가장 낮을 때에는 Overfitting되는 경우가 거의 대부분이기 때문에 Test Error가 가장 낮은 모델이 좋은 Classifier다.

Q2. (5점) 객관식

Linear regression 모델 구축하는 과정에서 어떤 coefficient 값이 매우 높은 음수값이 나왔다 (예: $\hat{\beta}_j = -157.321$). 다음 중 옳은 것을 고르고 이유를 설명하시오.

(a) 이 변수 X_j는 모델에 강한 영향을 미친다 (유지되어야 한다)

(b) 이 변수 X_j는 모델에 강한 영향을 미치지 않는다 (무시되어야 한다)

(c) 추가 정보 없이는 이 변수 X_j의 중요성에 대해 언급할 수 없다

(답): **(c)**.

$\hat{\beta}_j$의 **t-statistic**을 알 수 없어 **t-test hypothesis testing**을 통한 계수가 0이 아니라고 할 수 있는지 없는지 확인할 방법이 없다. 따라서, standard error, 전체 계수의 갯수, 전체 데이터의 갯수 등의 추가 정보 없이 X_j의 중요성에 대해 언급할 수 없다.

Q3. (10점) Overfitting Problem 에 대해 설명하고 해결 방안들에 대해 설명하시오.

(답): Overfitting Problem이란, 훈련데이터을 과도하게 학습하여 모델이 훈련데이터셋을 잘 맞추는 방향으로만 학습되어 훈련데이터가 아닌 새로운 데이터셋에서는 성능이 오히려 떨어지는 문제를 말한다. 이는 bias-variance trade-off와 같은 맥락이며, bias가 너무 작아져 variance가 커지는 경우이다. 보통 유연하고 복잡한 모델에서 발생된다.

이를 해결하기 위한 방안은 다음과 같은 방안들이 있다.

1. Multicollinearity 제거 등과 같은 Feature Selection을 통한 모델 복잡도 줄이기
2. L1, L2 Regularization 등의 규제 적용으로 bias를 높여 variance를 감소시키기
3. K-fold 교차검증과 같은 교차검증 기법 활용
4. Test Error가 줄어들다가 다시 증가하는 시점에 학습을 중단시키는 Early Stopping 기법 활용

Q4. (10점) Bayes Classifier에 대해 설명하고, Logistic Regression 과 KNN Classifier 의 관계에 대해 설명하시오.

(답):

Bayes Classifier이란,

bayes 정리에 기반하여 주어진 입력 x_0가 어떤 클래스 $Y \in C = \{1, 2, 3, \dots, K\}$에 속할 확률 $P(Y = j|X = X_0)$를 계산하여, 확률이 가장 큰 클래스로 분류하는 분류기다. 분포를 다 알고 있다는 가정 하에 Bayes classifier는 test error rate을 최소화시키는 분류기이지만, 실제 데이터의 모집단의 분포를 아는 경우는 없기 때문에 이론상으로만 가능한 이야기이기도 하다.

Logistic Regression과의 관계

Logistic Regression과의 공통점은 둘 다 Likelihood를 기반으로 추정을 한다는 점과 파라미터 추정을 통한 통계 모델 기반 분류기로써 설명력이 강하다는 점이 있다.

반대로 차이점은 Logistic Regression은 odds log값을 선형회귀하여 구하고, Bayes Classifier는 likelihood와 prior probability를 추정하여 분류한다는 점이 있다. 또한, Bayes Classifier는 모집단의 분포를 안다고 가정하는 반면, Logistic Regression은 로짓-선형 형태만 가정하고, 데이터를 통해 계수를 추정한다.

KNN과의 관계

KNN Classifier는 분포를 가정 또는 추정하지 않는 데이터 추정을 통한 모델이고, 데이터에 의존적이기 때문에 소수의 데이터 변동만 있어도 추정한 classifier의 변동이 크다. 반면, Bayes 분류기는 분

포의 가정, 추정을 통해 파라미터를 추정하는 방식으로 데이터의 변동에 강건한 특성을 지니고 있다.

Q5. Computing (50점)

- Download `UKM.csv` (training data) and `UKMtest.csv`
- 이 데이터는 학생들의 '전기 직류 기계(Electrical DC Machines)'에 대한 지식 상태에 관한 것입니다(Kahraman et al., 2013). 데이터셋 안에는 다음 6개의 변수가 있다.
- `STG` : 사용자의 목표 객체에 대한 학습 시간 정도
- `SCG` : 사용자의 목표 객체에 대한 반복 횟수 정도
- `STR` : 목표 객체와 관련된 항목에 대한 사용자의 학습 시간 정도
- `LPR` : 목표 객체와 관련된 항목에 대한 사용자의 시험 성적
- `PEG` : 목표 객체에 대한 사용자의 시험 성적
- `UNS` : 사용자의 지식 수준. Very Low(매우 낮음), Low(낮음), Middle(중간), High(높음)의 네 가지 클래스가 있습니다.

Goal:

- Predict `UNS` using all the other variables in the dataset.
- 이 데이터는 Test Data를 별도로 가지고 있다. Training Data를 통해 최대한 좋은 방법론을 찾아보자.

In [1]:
```
# Load the data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

ukm = pd.read_csv("UKM.csv")
ukmtest = pd.read_csv("UKMtest.csv")
```

Q5-1. Data 살펴보기

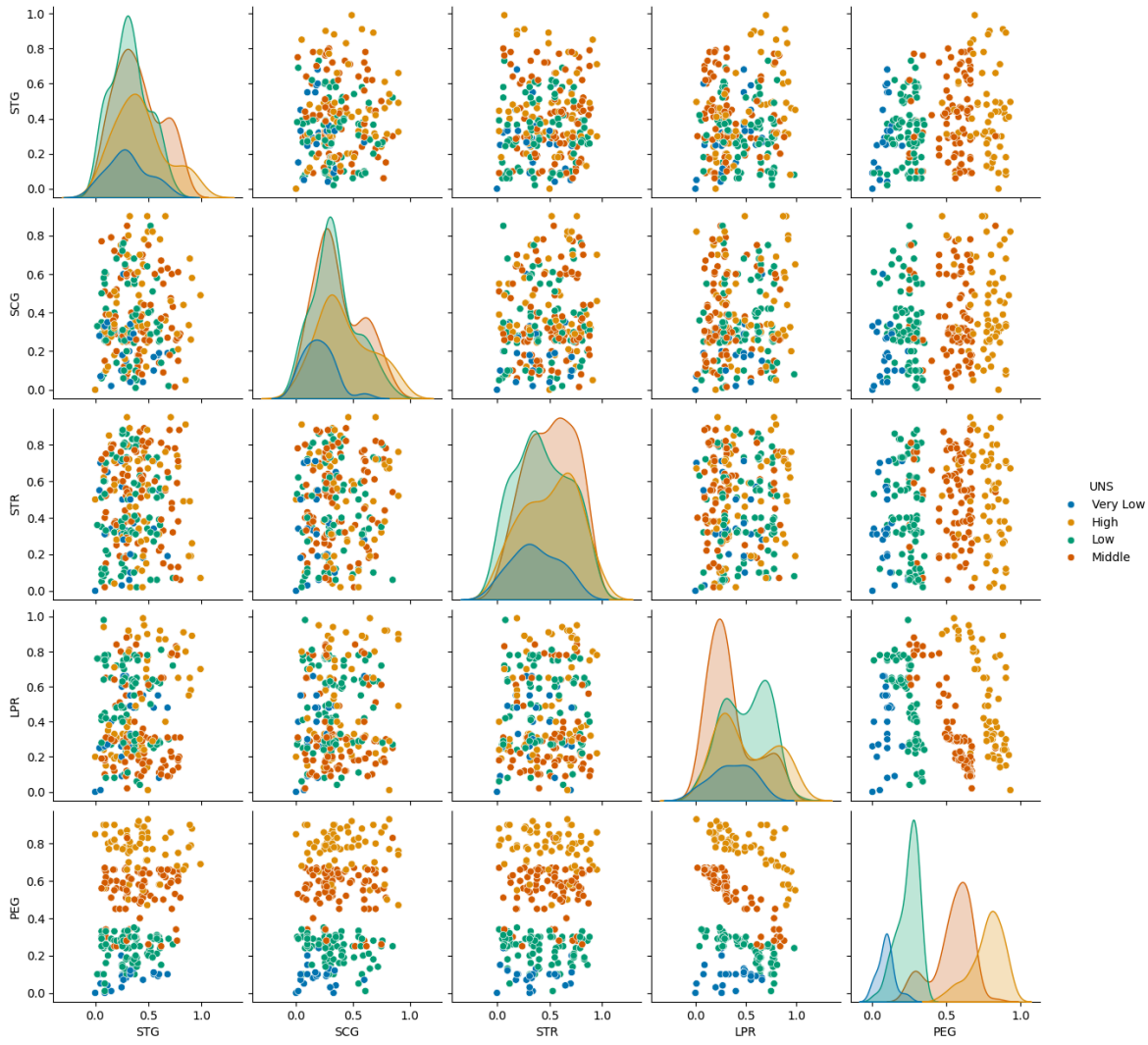
방법론 적용 전, Visualization, Summary 확인 등 EDA를 통해 데이터에 의미있는 패턴이 확인되는지 자유롭게 확인.

In [2]:
```
# EDA
display(ukm.head())
display(ukmtest.head())
```

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	Very Low
1	0.08	0.08	0.10	0.24	0.90	High
2	0.06	0.06	0.05	0.25	0.33	Low
3	0.10	0.10	0.15	0.65	0.30	Middle
4	0.08	0.08	0.08	0.98	0.24	Low

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.10	0.50	0.26	0.05	Very Low
1	0.05	0.05	0.55	0.60	0.14	Low
2	0.08	0.18	0.63	0.60	0.85	High
3	0.20	0.20	0.68	0.67	0.85	High
4	0.22	0.22	0.90	0.30	0.90	High

In [3]:
```
sns.pairplot(ukm, hue="UNS", height=2.5, palette="colorblind")
plt.show()
```

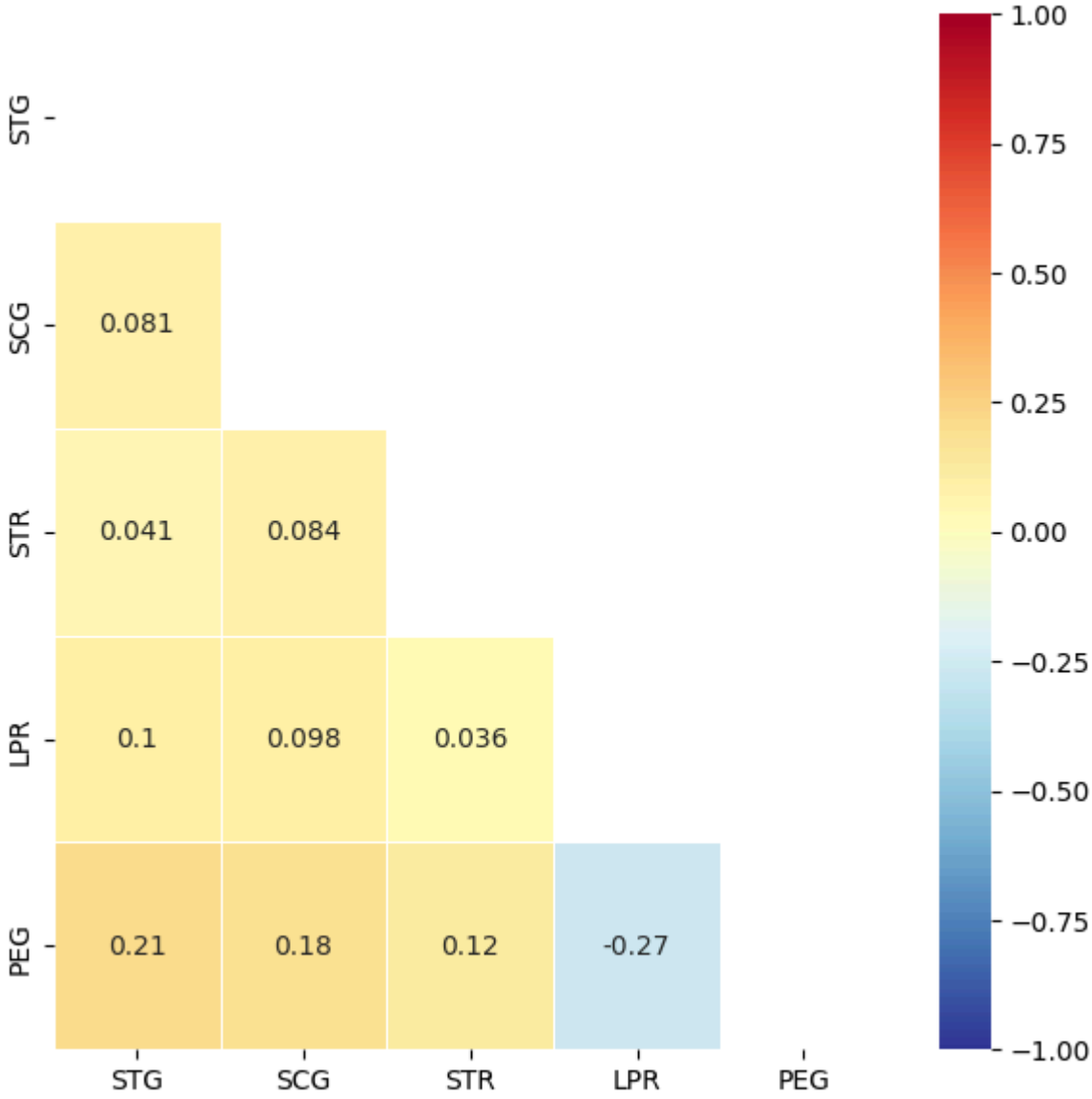


Pairplot 리뷰

PEG 컬럼에 따른 클래스 분류가 눈에 띄게 잘 되는 것을 볼 수 있다.

클래스별 히스토그램을 통해서도 그룹이 꽤 잘 구분되어 보이는 것으로 보아 PEG가 분류모델에서 가장 키가 되는 컬럼이 될 것으로 예상할 수 있다.

In [4]:
```
corr = ukm.drop(columns=['UNS']).corr()
# 그림 사이즈 지정
fig, ax = plt.subplots( figsize=(7,7) )

# 삼각형 마스크를 만든다(위 쪽 삼각형에 True, 아래 삼각형에 False)
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# 히트맵을 그린다
sns.heatmap(
    corr,
    cmap = 'RdYlBu_r',
    annot = True,   # 실제 값을 표시한다
    mask=mask,      # 표시하지 않을 마스크 부분을 지정한다
    linewidths=.5,  # 경계면 실선으로 구분하기
    vmin = -1,vmax = 1   # 컬러바 범위 -1 ~ 1
)
plt.show()
```



Multicollinearity 확인

Correlation Heatmap을 통해 독립변수 간의 correlation을 확인할 수 있다.

이때, 0.3을 넘는 값조차 없기 때문에 독립변수간의 다중공선성이 존재한다고 보기 힘들다.

따라서, 제거할만한 변수는 없기 때문에 모든 변수들을 모델 학습에 활용한다.

Split the data for KFold CV

K-fold CV를 활용해 여러 방법론을 비교할 때, 각 방법론마다 다른 group 이 설정되는 상황을 방지하기 위해 KFold 를 하나 만들어두고, 이 split 을 계속 이용하자.

Index를 뽑아서 manual 하게 CV Score들을 저장해도 되고, 이렇게 loading 된 `cv_DAS` 를, `GridSearchCV` 나 `cross_val_score` 등의 함수에 `cv` 라는 옵션에 사용해주면 된다.

In [5]:
```
from sklearn.model_selection import KFold
cv_DAS = KFold(n_splits = 5, random_state = 0, shuffle = True) # GridSear
```

Q5-2. KNN Classification

할 일

- Estimated test error를 가장 낮게 만드는 neighbors의 수 K를 찾기. (Grid for K: 1부터 25까지)
- Report the estimated test error rate with the best K.

In [6]:
```
# KNN Classification
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

le = LabelEncoder()
ss = StandardScaler()

X_train = ukm.drop(columns=['UNS'])
y_train = le.fit_transform(ukm['UNS'])

X_test = ukmtest.drop(columns=['UNS'])
y_test = le.transform(ukmtest['UNS'])

y_train
```

Out[6]:
```
array([3, 0, 1, 2, 1, 2, 2, 3, 1, 0, 0, 1, 2, 0, 2, 0, 3, 1, 1, 2, 1, 2,
       2, 1, 1, 0, 3, 1, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 3, 2, 1, 0, 2, 2,
       3, 2, 2, 0, 0, 2, 1, 2, 1, 0, 2, 1, 2, 2, 0, 1, 1, 2, 2, 1, 2, 3,
       3, 2, 1, 2, 2, 0, 3, 2, 0, 3, 1, 1, 3, 1, 1, 2, 2, 1, 1, 3, 0, 1,
       1, 2, 2, 2, 3, 1, 0, 1, 1, 0, 2, 1, 1, 0, 2, 1, 1, 2, 2, 2, 2, 0,
       1, 1, 0, 0, 2, 2, 3, 0, 2, 0, 1, 1, 1, 2, 0, 0, 0, 1, 1, 2, 1, 2,
       3, 0, 2, 2, 1, 1, 2, 1, 3, 2, 2, 1, 2, 0, 1, 3, 1, 0, 0, 1, 3, 1,
       2, 2, 2, 0, 2, 2, 1, 2, 0, 0, 3, 0, 2, 0, 1, 0, 1, 2, 0, 1, 2, 1,
       1, 0, 2, 0, 1, 0, 0, 1, 1, 1, 0, 2, 1, 0, 2, 0, 2, 0, 2, 1, 3, 1,
       2, 3, 3, 2, 3, 1, 0, 1, 2, 2, 3, 0, 2, 2, 0, 2, 2, 1, 1, 0, 2, 1,
       1, 2, 2, 2, 1, 0, 2, 1, 0, 0, 2, 1, 1, 0, 2, 1, 1, 0, 2, 2, 1, 0,
       2, 1, 2, 0, 2, 1, 0, 2, 2, 2, 1, 0, 2, 0, 2, 0])
```

In [7]:
```
le.inverse_transform([0, 1, 2, 3])
```

Out[7]:
```
array(['High', 'Low', 'Middle', 'Very Low'], dtype=object)
```

In [8]:
```
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

k_values = range(1, 25)
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled, y_train, scoring='accur
    # scores = cross_val_score(knn, X_train_scaled, y_train, scoring='rec
    cv_scores.append(scores.mean())


# 최적의 k 값 찾기
best_k = k_values[np.argmax(cv_scores)]
print("Best k:", best_k)

# 최적의 k 값으로 모델 피팅
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)
```

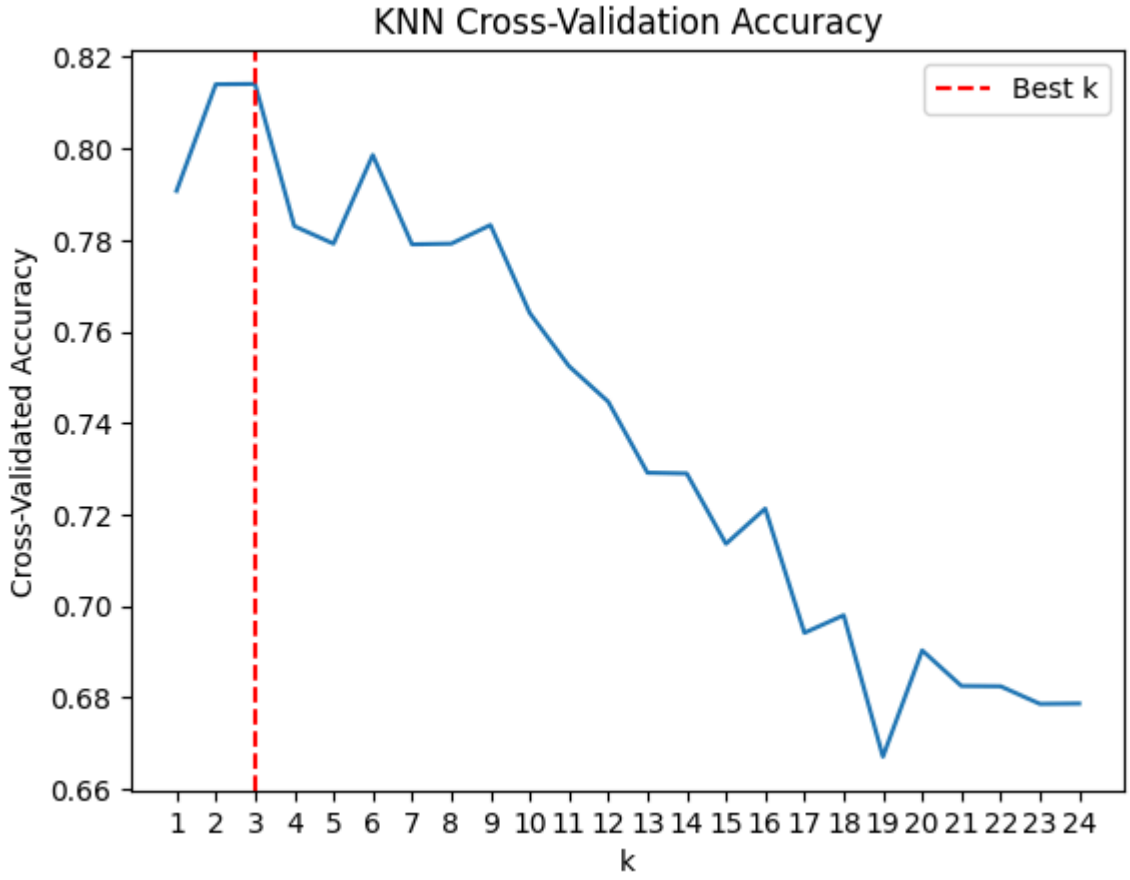
```
Best k: 3
```

Out[8]:

▾ KNeighborsClassifier

▶ Parameters

In [9]:
```
# 위의 scoring 들은 높을 수록 좋은 것 (loss 와 반대)
plt.plot(range(1,25), cv_scores)
plt.xlabel('k')
plt.ylabel('Cross-Validated Accuracy')
plt.title('KNN Cross-Validation Accuracy')
plt.xticks(range(1, 25))

# 최적의 k 값 위치 표시
plt.axvline(x=best_k, color='r', linestyle='--', label='Best k')
plt.legend()
plt.show()
```



In [10]:
```
from sklearn.metrics import confusion_matrix, classification_report
yhat_train = knn.predict(X_train_scaled)
cm = confusion_matrix(y_train, yhat_train)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[60  0  3  0]
 [ 0 83  0  0]
 [ 4  7 77  0]
 [ 0  8  0 16]]
```

In [11]:
```
print(classification_report(y_train, yhat_train, target_names=le.classes_
```

```
              precision    recall  f1-score   support

        High       0.94      0.95      0.94        63
         Low       0.85      1.00      0.92        83
      Middle       0.96      0.88      0.92        88
    Very Low       1.00      0.67      0.80        24

    accuracy                           0.91       258
   macro avg       0.94      0.87      0.89       258
weighted avg       0.92      0.91      0.91       258
```

In [12]:
```
test_accuracy_knn = max(cv_scores)
test_error_rate_knn = 1 - test_accuracy_knn
print("Test Accuracy with KNN:", test_accuracy_knn)
print("Estimated Test Error Rate:", test_error_rate_knn)
```

```
Test Accuracy with KNN: 0.8141025641025642
Estimated Test Error Rate: 0.1858974358974358
```

Q5-3. Logistic Regression

할 일

- penalty 가 없는 logistic regression 적용하기.
- 결과 해석 (추론).
- Report the estimated test error rate.

In [13]:
```
# Logistic Regression without penalty
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logit_class = LogisticRegression(penalty=None)

cv_scores_logit = []

# Training with cross-validation(ca_DAS)
for train_index, test_index in cv_DAS.split(X_train_scaled, y_train):
    X_train_cv, X_test_cv = X_train_scaled[train_index], X_train_scaled[t
    y_train_cv, y_test_cv = y_train[train_index], y_train[test_index]

    logit_class.fit(X_train_cv, y_train_cv)
    yhat = logit_class.predict(X_test_cv)
    cv_scores_logit.append(accuracy_score(y_test_cv, yhat))
```

```
/Users/cheoljunpark/ds_course/das511/.venv/lib/python3.11/site-packages/sk
learn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to c
onverge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/cheoljunpark/ds_course/das511/.venv/lib/python3.11/site-packages/sk
learn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to c
onverge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
```

Report the estimated test error rate

In [14]:
```
## - 결과 해석 (추론).
cv_scores_logit = np.array(cv_scores_logit)
test_accuracy_logit = cv_scores_logit.mean()
test_error_rate_logit = 1 - test_accuracy_logit
print("Test Accuracy with Logistic Regression:", test_accuracy_logit)
print("Estimated Test Error Rate:", test_error_rate_logit)
```

```
Test Accuracy with Logistic Regression: 0.9458521870286576
Estimated Test Error Rate: 0.05414781297134241
```

결과해석

- Test Error rate: 0.05415
- 모델의 성능은 전반적으로 우수한 분류 성능을 보임
- 설명력이 높은 Logistic Regression만으로도 좋은 성능을 보이기 때문에 복잡한 모델이 필요하지 않을 수 있음

Q5-4. SVM

할 일

- Gaussian RBF kernel 을 이용한 SVM 으로 classification
- `C` 와 `gamma` tuning. 최소 각각 2개 이상의 값에 대해 확인. 컴퓨팅이 여유있는 경우 gamma의 search grid 를 더 확장시키기.
- Report the estimated test error rate with the best hyperparameters.

In [15]:
```
from sklearn.svm import SVC

# SVM with Gaussian RBF Kernel
svm = SVC(kernel='rbf', gamma=1, C=10, decision_function_shape='ovo')
svm.fit(X_train_scaled, y_train)
yhat = svm.predict(X_test_scaled)
print("Confusion Matrix:\n", confusion_matrix(y_test, yhat))
print("Classification Report:\n")
print(classification_report(y_test, yhat, target_names=le.classes_))
```

```
Confusion Matrix:
 [[39  0  0  0]
 [ 0 39  6  1]
 [ 0  5 29  0]
 [ 0 13  0 13]]
Classification Report:

              precision    recall  f1-score   support

        High       1.00      1.00      1.00        39
         Low       0.68      0.85      0.76        46
      Middle       0.83      0.85      0.84        34
    Very Low       0.93      0.50      0.65        26

    accuracy                           0.83       145
   macro avg       0.86      0.80      0.81       145
weighted avg       0.85      0.83      0.82       145
```

In [16]:
```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(
    svm,
    {
        'C':[0.1,1,10,50,100,200,1000,10000],
        'gamma': np.logspace(-3, 2, 51)  # Using logspace to cover a rang
    },
    refit=True,
    cv=cv_DAS,
    scoring='accuracy'
)
grid.fit(X_train_scaled, y_train)
print("Best parameters found:")
print(grid.best_params_)

best_svm = grid.best_estimator_
test_accuracy_svm = grid.best_score_
test_error_rate_svm = 1 - test_accuracy_svm
print("\nTest Accuracy with Best SVM:", grid.best_score_)
print("\nEstimated Test Error Rate with Best SVM:", test_error_rate_svm)
```

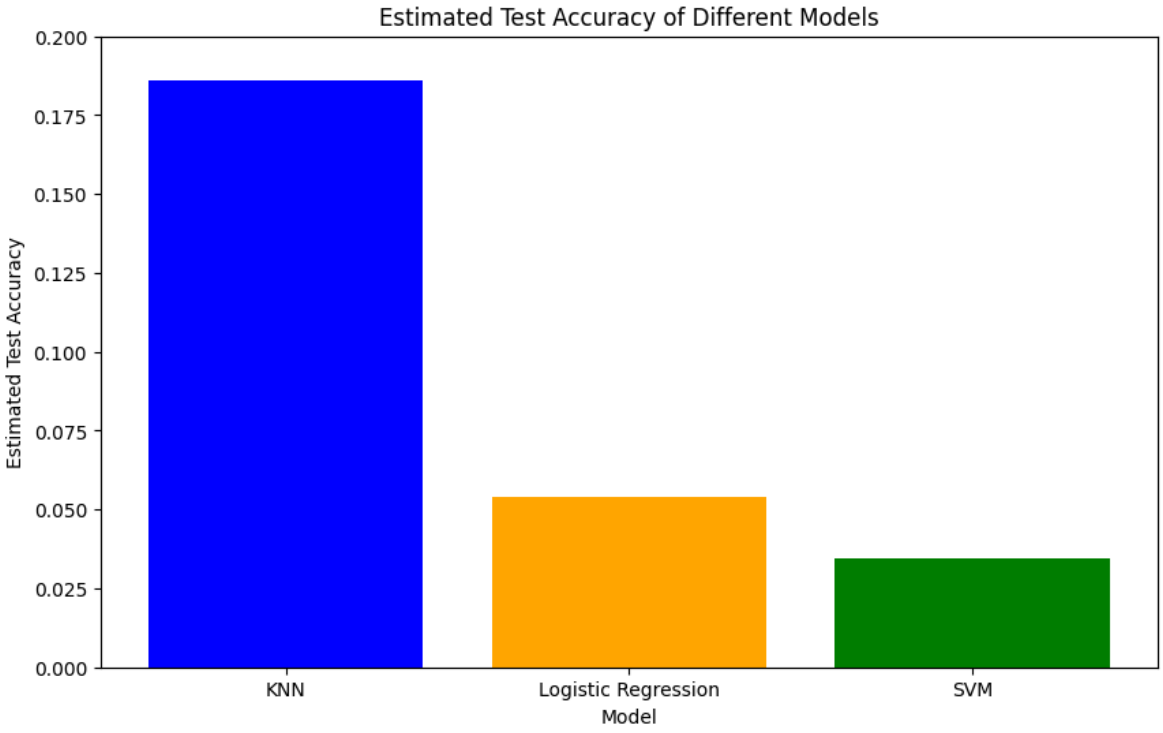
```
Best parameters found:
{'C': 100, 'gamma': np.float64(0.01)}

Test Accuracy with Best SVM: 0.9653092006033182

Estimated Test Error Rate with Best SVM: 0.03469079939668185
```

Q5-5. Conclusion with the Test Error rate

할 일

- Training data 를 기반으로 KNN, Logistic, SVM 방법론의 esitimated test error rate 비교 -> 제일 좋은 모델 report

In [17]:
```
# Compare the performance of the models
# Test 에러와 비교.
estimated_test_accuracy = {
    "KNN": test_error_rate_knn,
    "Logistic Regression": test_error_rate_logit,
    "SVM": test_error_rate_svm,
}
plt.figure(figsize=(10, 6))
plt.bar(estimated_test_accuracy.keys(), estimated_test_accuracy.values(),
plt.xlabel('Model')
plt.ylabel('Estimated Test Accuracy')
plt.title('Estimated Test Accuracy of Different Models')
plt.ylim(0, 0.2)
plt.show()
```



- 선택된 hyperparmeter를 그대로 활용해서 전체 training data fit
- Predict the test data (`UKMtest.csv`) and report error rates (KNN, Logistic, SVM)

In [18]:
```
tuned_svm = SVC(kernel='rbf', decision_function_shape='ovo', **grid.best_
tuned_svm.fit(X_train_scaled, y_train)
yhat_svm = tuned_svm.predict(X_test_scaled)
test_error_rate_svm2 = 1 - accuracy_score(y_test, yhat_svm)
print("Test Error Rate with Tuned SVM:", test_error_rate_svm2)
print("\nConfusion Matrix:\n", confusion_matrix(y_test, yhat_svm))
print("\nClassification Report:")
print(classification_report(y_test, yhat_svm, target_names=le.classes_))
```

```
Test Error Rate with Tuned SVM: 0.03448275862068961

Confusion Matrix:
 [[39  0  0  0]
 [ 0 46  0  0]
 [ 0  2 32  0]
 [ 0  3  0 23]]

Classification Report:
              precision    recall  f1-score   support

        High       1.00      1.00      1.00        39
         Low       0.90      1.00      0.95        46
      Middle       1.00      0.94      0.97        34
    Very Low       1.00      0.88      0.94        26

    accuracy                           0.97       145
   macro avg       0.98      0.96      0.96       145
weighted avg       0.97      0.97      0.97       145
```

In [19]:
```
tuned_logit = LogisticRegression(penalty=None)
tuned_logit.fit(X_train_scaled, y_train)
yhat_logit = tuned_logit.predict(X_test_scaled)
test_error_rate_logit2 = 1 - accuracy_score(y_test, yhat_logit)
print("Test Error Rate with Tuned Logistic Regression:", test_error_rate_
print("\nConfusion Matrix:\n", confusion_matrix(y_test, yhat_logit))
print("\nClassification Report:")
print(classification_report(y_test, yhat_logit, target_names=le.classes_)
```

```
Test Error Rate with Tuned Logistic Regression: 0.04137931034482756

Confusion Matrix:
 [[39  0  0  0]
 [ 0 43  2  1]
 [ 2  1 31  0]
 [ 0  0  0 26]]

Classification Report:
              precision    recall  f1-score   support

        High       0.95      1.00      0.97        39
         Low       0.98      0.93      0.96        46
      Middle       0.94      0.91      0.93        34
    Very Low       0.96      1.00      0.98        26

    accuracy                           0.96       145
   macro avg       0.96      0.96      0.96       145
weighted avg       0.96      0.96      0.96       145
```

In [20]:
```
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)
yhat_knn = knn.predict(X_test_scaled)
test_error_rate_knn2 = 1 - accuracy_score(y_test, yhat_knn)
print("Test Error Rate with Tuned KNN:", test_error_rate_knn2)
print("\nConfusion Matrix:\n", confusion_matrix(y_test, yhat_knn))
print("\nClassification Report:")
print(classification_report(y_test, yhat_knn, target_names=le.classes_))
```

```
Test Error Rate with Tuned KNN: 0.17931034482758623

Confusion Matrix:
 [[36  0  3  0]
 [ 0 42  2  2]
 [ 2  6 26  0]
 [ 0 11  0 15]]

Classification Report:
              precision    recall  f1-score   support

        High       0.95      0.92      0.94        39
         Low       0.71      0.91      0.80        46
      Middle       0.84      0.76      0.80        34
    Very Low       0.88      0.58      0.70        26

    accuracy                           0.82       145
   macro avg       0.85      0.79      0.81       145
weighted avg       0.84      0.82      0.82       145
```

- Comment any conclusion

(결론):

특정 한 컬럼에 의해서 분류가 꽤 잘되는 데이터의 경우, KNN 분류기보다 Logistic Regression 분류기(이하 Logit)나 Support Vector Mechine 분류기(이하 SVM)가 더 성능이 좋다는 점을 유추할 수 있음.

각 클래스별 precision, recall, f1 score 모두 높은 값을 가지는 것으로 보아 전반적으로 클래스간 균형이 잘 잡혀있는 것으로 보여 SVM, Logit 모델의 성능은 충분히 좋다고 볼 수 있음.

또한 SVM, Logit 모델의 test error rate이 큰 차이 없이 0.05 미만이라 설명력이 강한 Logit 모델을 활용하는 것으로도 충분할 것으로 보임.

따라서, Logit 모델을 해당 데이터에 대한 최적의 분류 모델로 선정함.