GPU

3조 김주현

목차

Index.

PART 1.

문제 정의

* 성능에 따른 GPU 가격 예측

* 성능에 따른 GPU 필요 전력량 확인

문제 정의

	Architecture	Best_Resolution	Boost_Clock	Core_Spee	DVI_Conne	Dedicated	Direct_X	DisplayPor	HDMI_Cor	Integrated	L2_Cache	Manufactu	Max_Powe	Memory	Memory_E	Memory_E	Memory_S
2	Tesla G92b			738 MHz	2	Yes	DX 10.0		0	No	0KB	Nvidia	141 Watts	1024 MB	64GB/sec	256 Bit	1000 MHz
3	R600 XT	1366 x 768		-	2	Yes	DX 10		0	No	0KB	AMD	215 Watts	512 MB	106GB/sec	512 Bit	828 MHz
4	R600 PRO	1366 x 768		-	2	Yes	DX 10		0	No	0KB	AMD	200 Watts	512 MB	51.2GB/se	256 Bit	800 MHz

전체 데이터 3344개 중, '출시당시 가격' 557개 (약 17%) => 쓸 수 없음

GPU 성능에 따른 필요 전력량을 분류해봄으로서, 원하는 GPU와의 전력량 비교

전체 데이터 3344개 중, '필요 전력량' 2782개 (약 83%) => 쓸 수 있음

PART 2.

데이터 전처리

데이터 전처리

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2013 entries, 0 to 2012
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Name              2013 non-null   object
 1   Core_Speed        2013 non-null   int64
 2   Direct_X          2013 non-null   float64
 3   Integrated        2013 non-null   object
 4   L2_Cache          2013 non-null   int64
 5   Manufacturer      2013 non-null   object
 6   Max_Power         2013 non-null   int64
 7   Memory            2013 non-null   int64
 8   Memory_Bandwidth  2013 non-null   float64
 9   Memory_Bus        2013 non-null   int64
 10  Memory_Speed      2013 non-null   int64
 11  Memory_Type       2013 non-null   object
 12  Open_GL           2013 non-null   float64
 13  Pixel_Rate        2013 non-null   int64
 14  Process           2013 non-null   int64
 15  ROPs              2013 non-null   int64
 16  Resolution_WxH    2013 non-null   object
 17  SLI_Crossfire     2013 non-null   object
 18  Shader            2013 non-null   float64
 19  TMUs              2013 non-null   int64
 20  Texture_Rate      2013 non-null   int64
 21  RMax_Power        2013 non-null   int64
dtypes: float64(4), int64(12), object(6)
memory usage: 346.1+ KB
```

전체 컬럼 34개 중,
null 개수가 비교적 많은 컬럼은 삭제하고
22개 컬럼만 살려둠

데이터 전처리

DX 10

DX 10.1

DX 10

DX 11.2

```
[ ]   1 df['Direct_X'] = df['Direct_X'].astype('int64')
      2 df['Direct_X'] = df['Direct_X'].astype('object')
```

```
[ ]   1 df['Direct_X'].unique()
```

```
array([10, 11, 12, 9], dtype=object)
```

Core_Speed	MHz
L2_Cache	KB
Max_Power	Watts
Memory	MB
Memory_Bandwidth	GB/sec
Memory_Bus	Bit
Memory_Speed	MHz
Pixel_Rate	GPixel/s
Process	nm
Texture_Rate	GTexel/s

'Direct_X' 에서 'DX' 를 제거한 뒤,
int형으로 변환하여 소수점을 날리고
원-핫코딩을 위해 object로 다시 변환

마찬가지로, 다른 컬럼의 단위도 모두 제거

데이터 전처리

Max_Powe
141 Watts
215 Watts
200 Watts
45 Watts
50 Watts
190 Watts

=ROUNDUP(G2,-2)

RMax_Power
200
300
200
100
100
200
200

파워 서플라이를 구매할 때,
보통 100W 단위로 업그레이드하기 때문에 10의 자리에서 올림해줌.

PART 3.

적합한 모델 분석

적합한 모델 분석

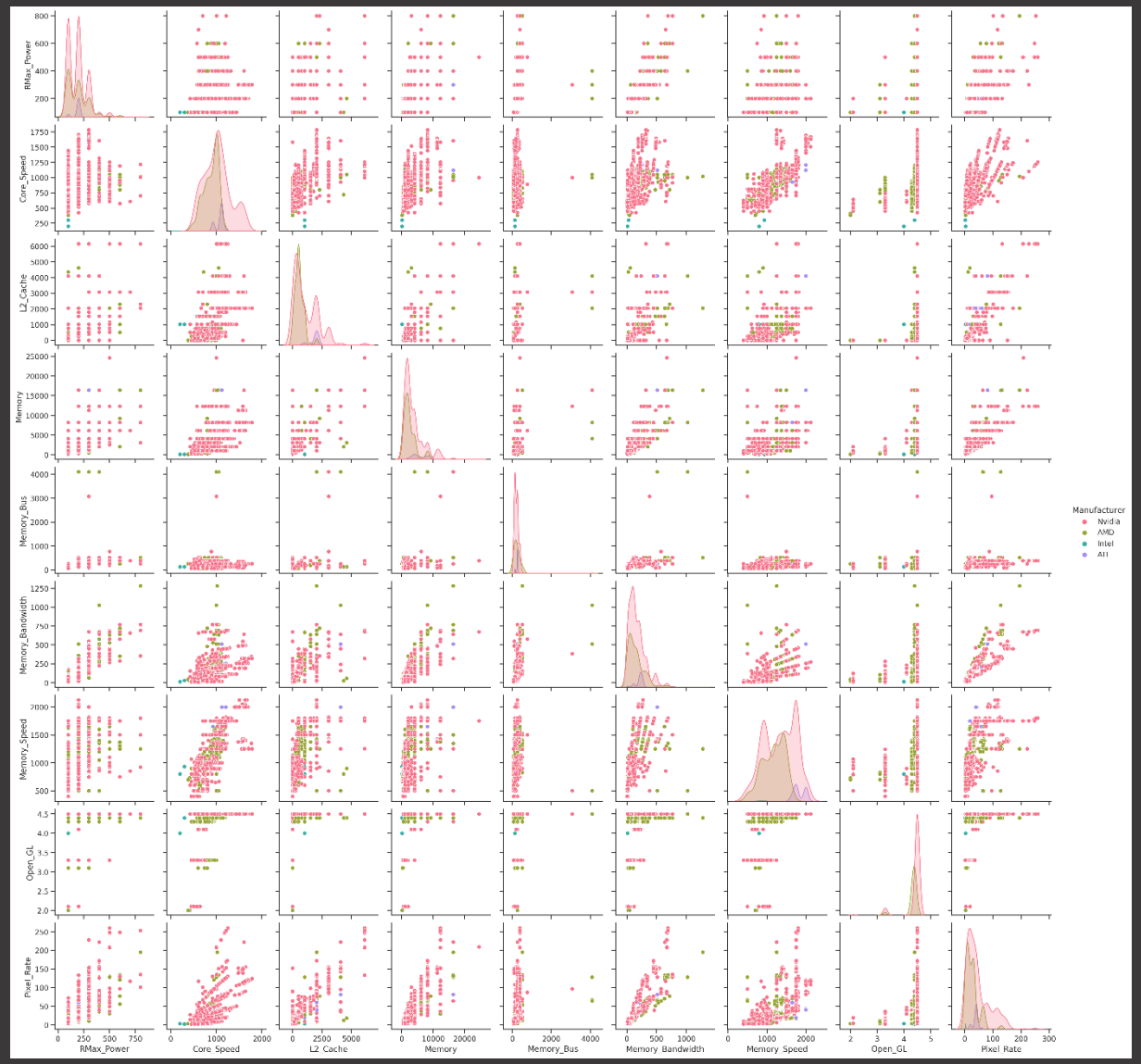
```
onehot_in = pandas.get_dummies(df[['Direct_X', 'Integrated', 'Manufacturer', 'Memory_Type', 'SLI_Crossfire']])

df = pandas.concat([df, onehot_in], axis = 1)

df.drop(['Name', 'Direct_X', 'Integrated', 'Manufacturer', 'Memory_Type', 'SLI_Crossfire', 'Max_Power'], axis = 1, inplace = True)

X = df[['Core_Speed', 'L2_Cache', 'Memory', 'Memory_Bandwidth', 'Memory_Bus', 'Memory_Speed', 'Open_GL',  'Pixel_Rate', 'Process',
y = df['RMax_Power']
```

[5]
```
from sklearn import preprocessing

X = preprocessing.StandardScaler().fit(X).transform(X)
```

[6]
```
from sklearn.model_selection import  train_test_split

X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.3, random_state=777) # 랜덤 추출 값
```

원-핫코딩 후 데이터를 분리

적합한 모델 분석




```
[ ]   1 import seaborn as sns
      2 #del data['Id'] # 인덱스 열 삭제
      3 sns.set(style="ticks", color_codes=True)
      4 g = sns.pairplot(df, hue="Manufacturer", palette="husl")
```

변수끼리의 관계 시각화해본 결과,
NVIDIA 제품이 압도적으로 많아
확인하기가 힘듦

적합한 모델 분석_Logistic Regression

머신러닝 모델 구축 - 1) LogisticRegression

```
[8]   1 log_reg = LogisticRegression().fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or 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-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
[9]   1 # 정확도 확인
      2 print('Train data Score: ', log_reg.score(X_train, y_train))
      3 print('Test data Score: ', log_reg.score(X_test, y_test))
```

```
Train data Score:  0.8573456352022711
Test data Score:  0.8559602649006622
```

적합한 모델 분석_Decision Tree

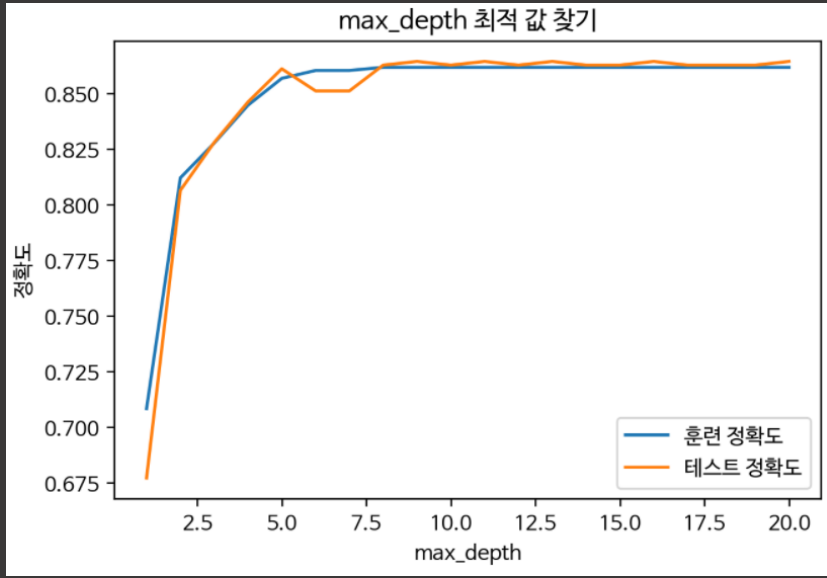
머신러닝 모델 구축 - 2) Decision Tree

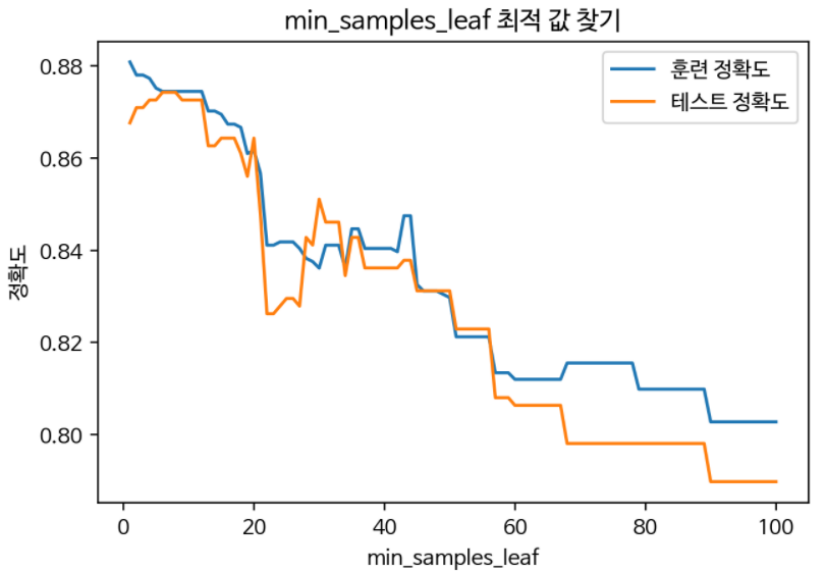
```
[10]  1 tree = DecisionTreeClassifier(max_depth = 10,
      2                               min_samples_leaf = 20,
      3                               min_samples_split = 40).fit(X_train, y_train)
```

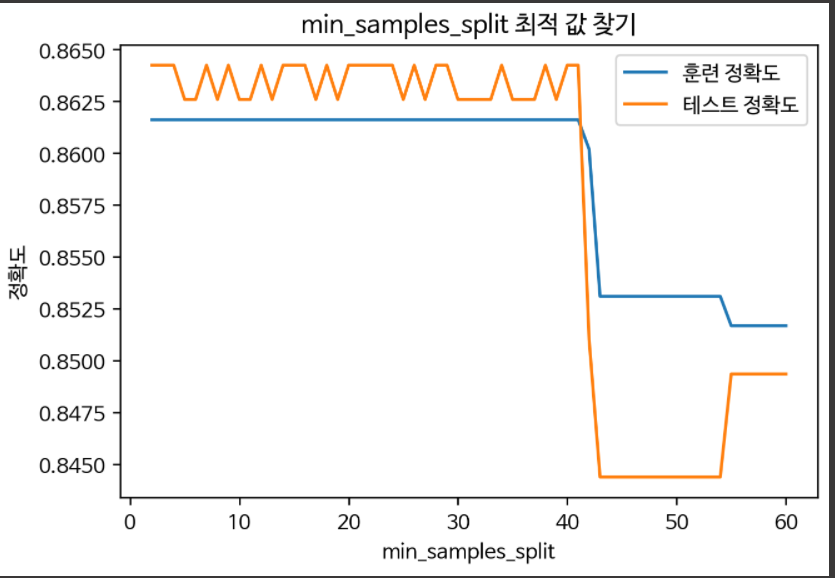
```
[11]  1 # 정확도 확인
      2 print('Train data Score: ', tree.score(X_train, y_train))
      3 print('Test data Score: ', tree.score(X_test, y_test))
```

```
Train data Score:  0.8616039744499645
Test data Score:  0.8642384105960265
```

적합한 모델 분석_Decision Tree_최적 값 찾기







적합한 모델 분석_Gradient Boosting Classifier

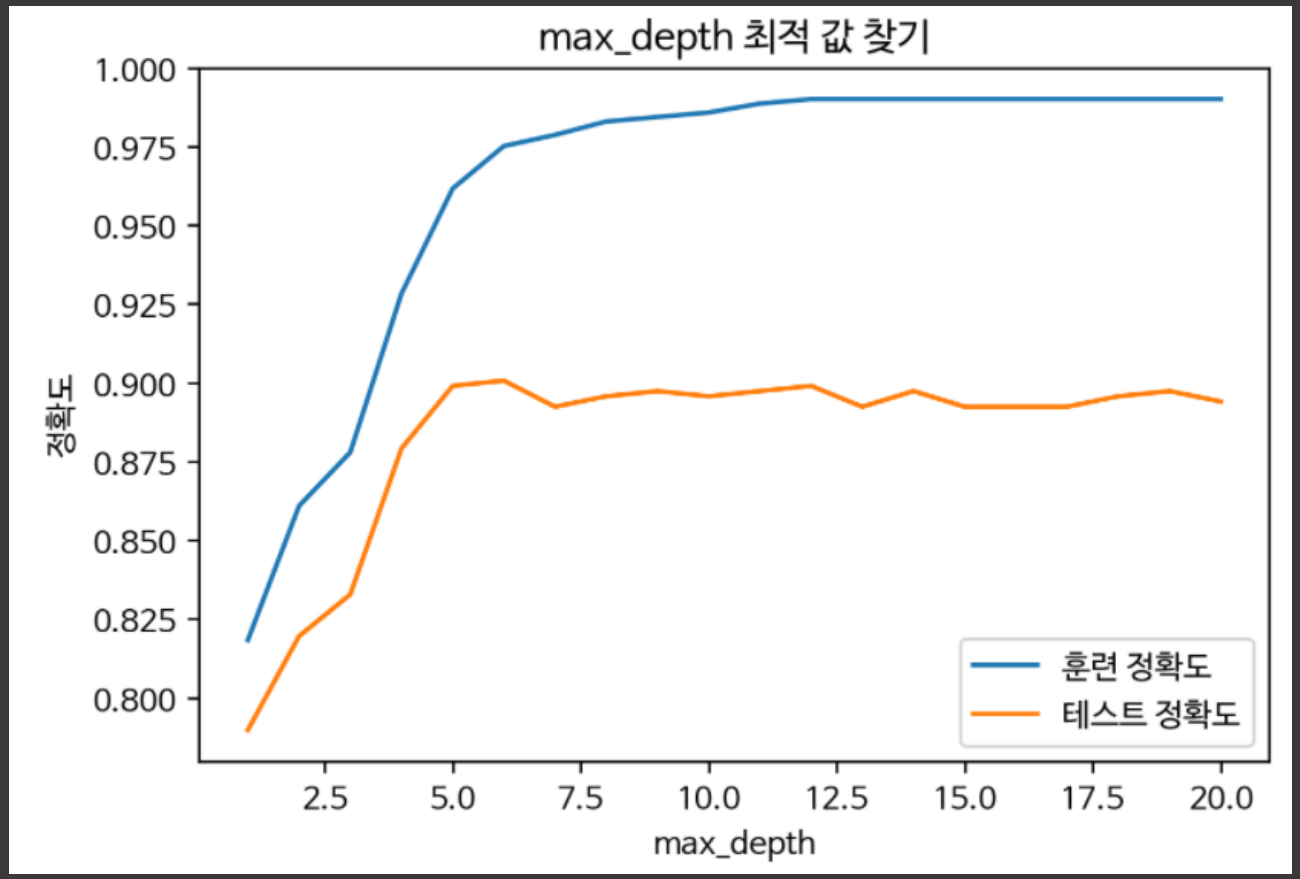
머신러닝 모델 구축 - 3) GradientBoostingClassifier

```
[15]  1 boost = GradientBoostingClassifier(max_depth = 5,
      2                                    learning_rate = 0.01).fit(X_train, y_train)
```

```
[16]  1 # 정확도 확인
      2 print('Train data Score: ', boost.score(X_train, y_train))
      3 print('Test data Score: ', boost.score(X_test, y_test))
```

```
Train data Score:  0.9616749467707594
Test data Score:  0.8990066225165563
```

적합한 모델 분석_Gradient Boosting Classifier_최적 값 찾기



적합한 모델 분석_Random Forest Classifier

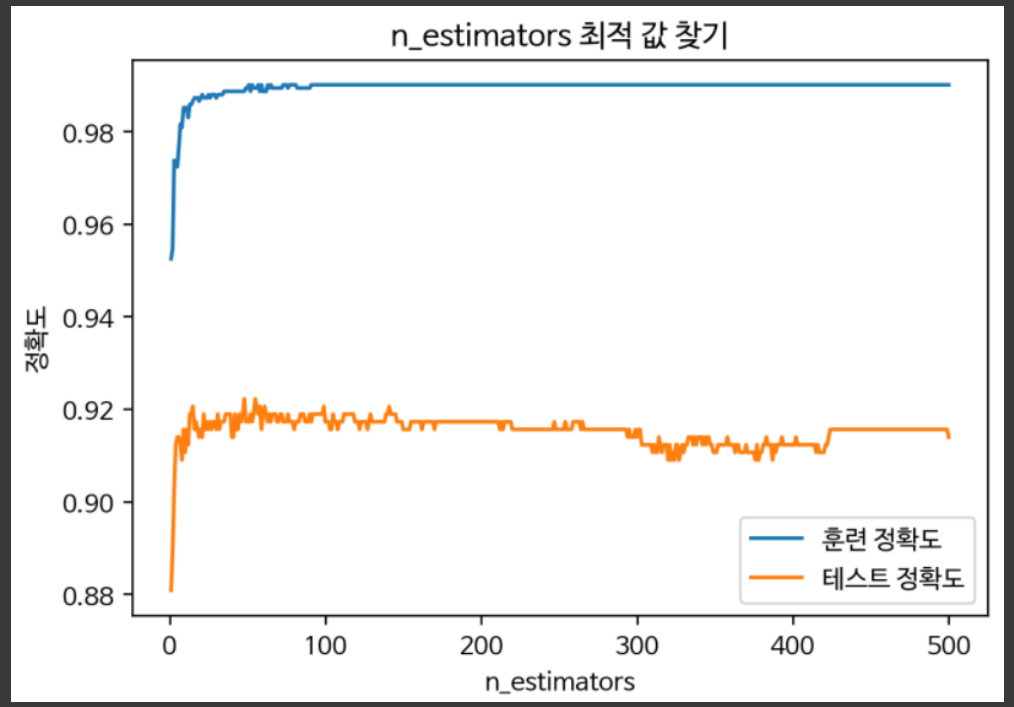
머신러닝 모델 구축 - 4) RandomForestClassifier

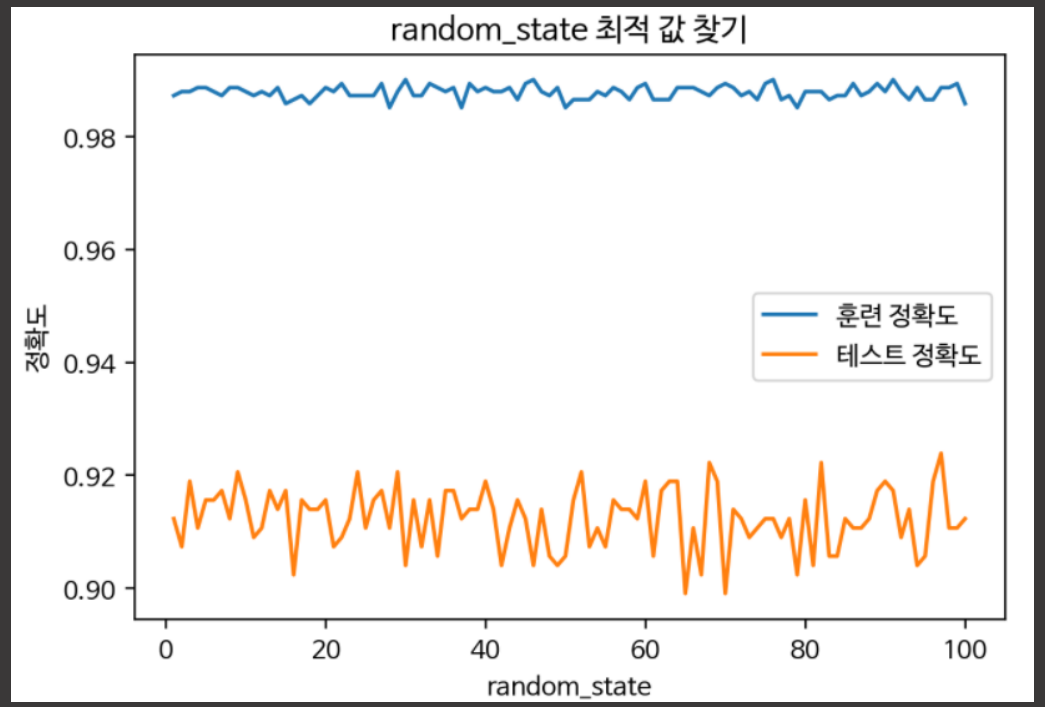
```
[18]  1 random = RandomForestClassifier(n_estimators = 300,
      2                                 random_state = 0).fit(X_train, y_train)
```

```
[19]  1 # 정확도 확인
      2 print('Train data Score: ', random.score(X_train, y_train))
      3 print('Test data Score: ', random.score(X_test, y_test))
```

```
Train data Score:  0.9900638750887154
Test data Score:  0.9139072847682119
```

적합한 모델 분석_Random Forest Classifier_최적 값 찾기





적합한 모델 분석_평가지표 분석

Logistic Regression 평가 지표

	precision	recall	f1-score	support
100	0.94	0.92	0.93	250
200	0.80	0.88	0.84	224
300	0.78	0.70	0.74	98
400	0.38	0.19	0.25	16
500	0.38	0.43	0.40	7
600	0.67	0.57	0.62	7
800	0.00	0.00	0.00	2
accuracy			0.84	604
macro avg	0.56	0.53	0.54	604
weighted avg	0.83	0.84	0.84	604

Decision Tree 평가 지표

	precision	recall	f1-score	support
100	0.90	0.94	0.92	250
200	0.83	0.79	0.81	224
300	0.70	0.83	0.76	98
400	0.00	0.00	0.00	16
500	0.29	0.71	0.42	7
600	0.00	0.00	0.00	7
800	0.00	0.00	0.00	2
accuracy			0.82	604
macro avg	0.39	0.47	0.42	604
weighted avg	0.80	0.82	0.81	604

GradientBoostingClassifie 평가 지표

	precision	recall	f1-score	support
100	0.98	0.95	0.97	250
200	0.88	0.93	0.90	224
300	0.82	0.85	0.83	98
400	0.33	0.19	0.24	16
500	0.71	0.71	0.71	7
600	0.50	0.43	0.46	7
800	0.00	0.00	0.00	2
accuracy			0.90	604
macro avg	0.60	0.58	0.59	604
weighted avg	0.89	0.90	0.89	604

RandomForestClassifier 평가 지표

	precision	recall	f1-score	support
100	0.97	0.98	0.97	251
200	0.91	0.95	0.93	207
300	0.90	0.88	0.89	112
400	0.50	0.35	0.41	17
500	0.82	0.90	0.86	10
600	0.50	0.33	0.40	6
800	0.00	0.00	0.00	1
accuracy			0.92	604
macro avg	0.66	0.63	0.64	604
weighted avg	0.92	0.92	0.92	604

적합한 모델 분석

```
[ ]  1 onehot_in = pandas.get_dummies(df[['Direct_X', 'Integrated', 'Manufacturer', 'Memory_Type', 'SLI_Crossfire']])
     1 df = pandas.concat([df, onehot_in], axis = 1)
```

Direct_X_9	Direct_X_10	Direct_X_11	Direct_X_12	Integrated_No	Integrated_Yes	Manufacturer_AMD

```
1 # 기존 컬럼 삭제
2 df.drop(['Name', 'Direct_X', 'Integrated', 'Manufacturer', 'Memory_Type', 'SLI_Crossfire', 'Max_Power'], axis = 1, inplace = True)
3 df.head(3)
```

Core_Speed	L2_Cache	Memory	Memory_Bandwidth	Memory_Bus	Memory_Speed	Open_GL	Pixel_Rate	Process	ROPs	Resolution_WxH

Object형을 원-핫코딩 해주고 기존 컬럼을 삭제

적합한 모델 분석

```
1 from sklearn import preprocessing
2
3 X = preprocessing.StandardScaler().fit(X).transform(X)
```

컬럼 간 수의 차가 심해, 정규화 시켜줌

```
4 from sklearn import preprocessing
5
6 label_encoder = preprocessing.LabelEncoder()     # label encoder 생성
7
8 onehot_location = label_encoder.fit_transform(df['RMax_Power'])
9
10 df['y_label'] = onehot_location
11
```

```
5 from tensorflow.keras.utils import  to_categorical
6
7 #각 데이터의 레이블 0~9 숫자 값을 범주형 형태로 변경
8
9 y = to_categorical(y)
10 y
```

```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

다중 분류이기 때문에
label encode를 사용함

적합한 모델 분석

```
[8]  1 from sklearn.model_selection import  train_test_split
     2
     3 X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.3, random_state=777)
```

훈련 데이터 셋, 테스트 데이터 셋 생성

```
[10]  1 from sklearn.model_selection import  train_test_split
      2
      3 X_train, X_val, y_train, y_val = train_test_split(X_train,          # 독립 변수
      4                                                   y_train,          # 종속변수
      5                                                   test_size=0.3,    # 검증 33%
      6                                                   random_state=777)   # 랜덤 추출 값
```

```
[11]  1 print(f'훈련 데이터 {X_train.shape} 레이블 {y_train.shape}')
      2 print(f'검증 데이터 {X_val.shape} 레이블 {y_val.shape}')
```

```
훈련 데이터 (986, 36) 레이블 (986, 8)
검증 데이터 (423, 36) 레이블 (423, 8)
```

검증 데이터 셋 생성

적합한 모델 분석_훈련 검증 정확도, 손실 값 그래프

[12]
```
1 import numpy as np
2 np.set_printoptions(suppress=True)
3
4 print(X_train[0])
```

```
[ 0.02438634 -0.21744566  0.28166656  0.88311432 -0.13126993  0.53311984
  0.37512859  0.28970644 -0.24805475  0.21457216  0.25314938  0.06040245
  0.71573087 -0.03153617 -0.26167028 -0.32283712  0.43556436  0.03863337
 -0.03863337 -0.70315864 -0.19259876 -0.03153617  0.76243295 -0.0222939
 -0.09761448 -0.42827738 -0.03863337 -0.2250029  -0.03863337  0.59092784
 -0.2007737  -0.0499003  -0.03153617 -0.0222939  -0.6984303   0.6984303 ]
```

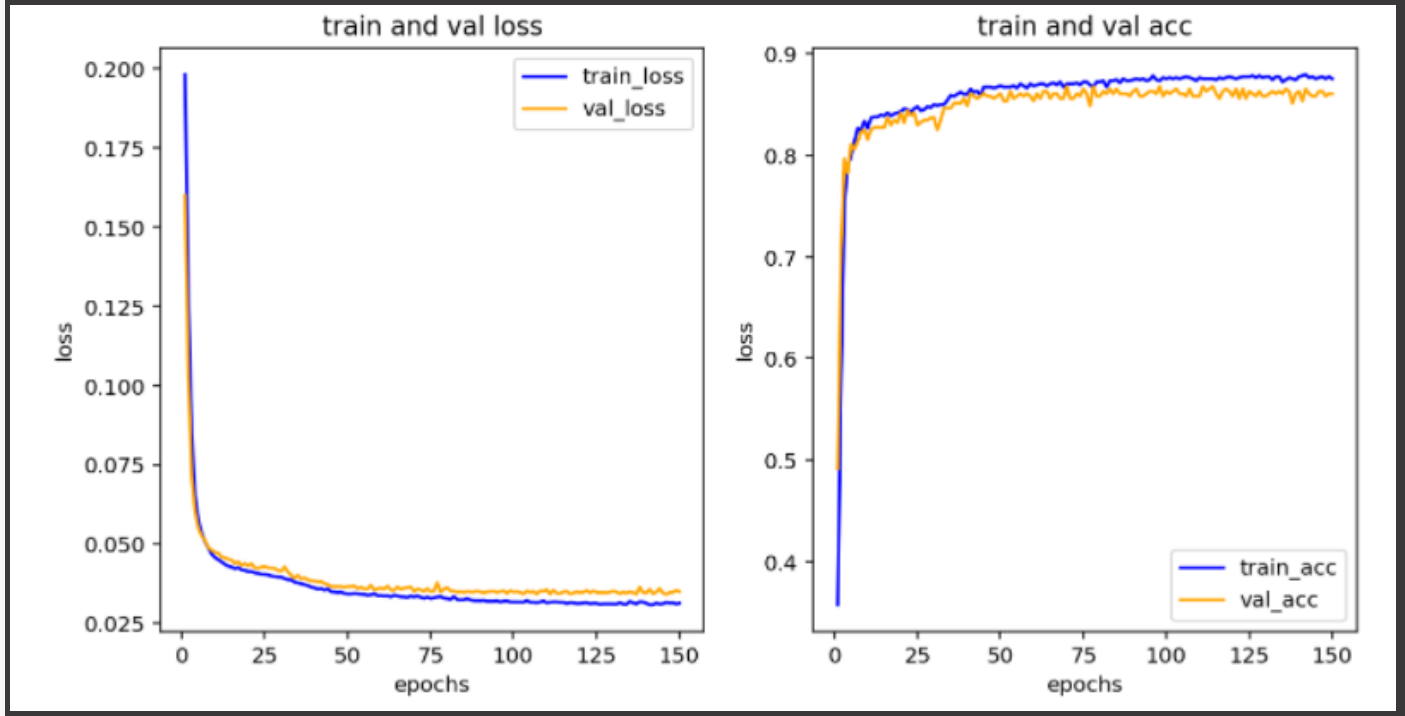
[14]
```
1 from keras.models import Sequential
2 from keras.layers import Dense
3
4 model = Sequential()
5 model.add(Dense(64, activation = 'relu', input_shape = (36, )))
6 model.add(Dense(32, activation = 'relu'))
7 model.add(Dense(8,activation = 'softmax'))  # 다중분류에서 가장 확률이 높은 하나 값만 가져오기위해서 'softmax' 사용
```

[15]
```
1 model.compile(loss='mae',
2                optimizer = 'adam',
3                metrics =['acc'] ) # 정확도
```

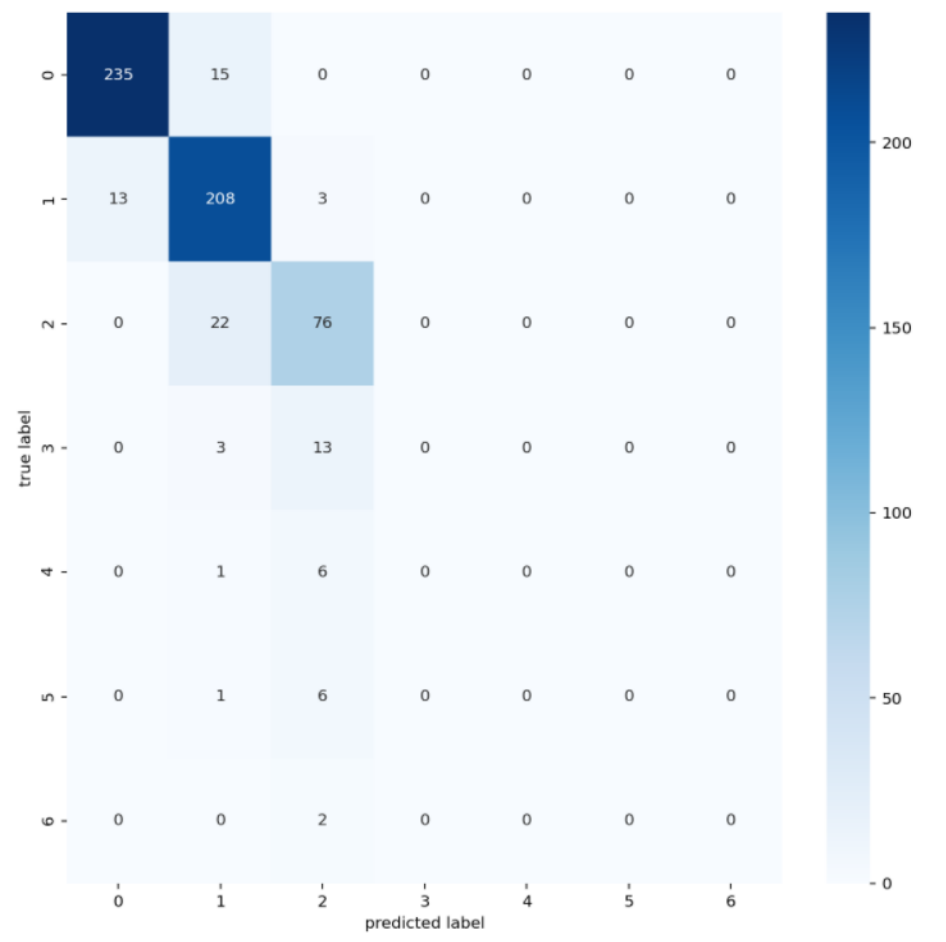
[16]
```
1 history = model.fit(X_train,y_train,
2                     epochs = 150,
3                     validation_data =(X_val,y_val))
```

적합한 모델 분석_훈련 검증 정확도, 손실 값 그래프

```
[17]  1 import matplotlib.pyplot as plt
      2
      3 his_dict = history.history
      4 loss = his_dict['loss']
      5 val_loss = his_dict['val_loss'] # 검증 데이터가 있는 경우 'val_' 수식어가 붙습니다.
      6
      7 epochs = range(1, len(loss) + 1)
      8 fig = plt.figure(figsize = (10, 5))
      9
     10 # 훈련 및 검증 손실 그리기
     11 ax1 = fig.add_subplot(1, 2, 1)
     12 ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
     13 ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
     14 ax1.set_title('train and val loss')
     15 ax1.set_xlabel('epochs')
     16 ax1.set_ylabel('loss')
     17 ax1.legend()
     18
     19 acc = his_dict['acc']
     20 val_acc = his_dict['val_acc']
     21
     22 # 훈련 및 검증 정확도 그리기
     23 ax2 = fig.add_subplot(1, 2, 2)
     24 ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
     25 ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
     26 ax2.set_title('train and val acc')
     27 ax2.set_xlabel('epochs')
     28 ax2.set_ylabel('loss')
     29 ax2.legend()
     30
     31 plt.show() # loss 값이 손실 값이고 정답률 높은데 오답률은 줄어들고 2 정확도가
```



적합한 모델 분석_혼동 행렬 그래프




```python
# sklearn.metrics 모듈은 여러가지 평가 지표에 관한 기능을 제공합니다.
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 혼동 행렬을 만듭니다.
plt.figure(figsize = (10, 10))
cm = confusion_matrix(np.argmax(y_test, axis = -1), np.argmax(results, axis = -1))
sns.heatmap(cm, annot = True, fmt = 'd',cmap = 'Blues')
plt.xlabel('predicted label')
plt.ylabel('true label')
plt.show()
```

y_label	RMax_Power
0	100
1	200
2	300
3	400
4	500
5	600
6	700
7	800

PART 4.

시사점

시사점

학습한 역대 GPU 성능별 전력량과
비교하여 적당한지 판별

PART 5.

한계점 및 소감

데이터 부족

출시 당시 가격 null 값,
많은 null 값,
최신 데이터의 부재

모델의 이해

분석 목적과
데이터 유형에 따른 전처리 중요,
모델의 옵션 이해 부족

감사합니다