```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```python
!ls "/content/drive/My Drive"
```

```
'Colab Notebooks'  'Fixed Resume.pdf'  'M.Tech DA'   Resume
Resume1.pdf
```

```python
import pandas as pd
data=pd.read_csv("/content/drive/My Drive/Colab
Notebooks/nursery.csv")
data
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 12960,\n
\"fields\": [\n    {\n      \"column\": \"parents\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"usual\",\n          \"pretentious\",\n          \"great_pret\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"has_nurs\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"less_proper\",\n          \"very_crit\",\n          \"improper\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"form\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          \"completed\",\n
\"foster\",\n          \"complete\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"children\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          \"2\",\n          \"more\",\n
\"1\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"housing\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"convenient\",\n          \"less_conv\",\n
\"critical\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"finance\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"inconv\",\n          \"convenient\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"social\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"nonprob\",\n
\"slightly_prob\"\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"health\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"recommended\",\n   \"priority\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"final evaluation\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"priority\",\n   \"spec_prior\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"data"}

```
data.columns
```

```
Index(['parents', 'has_nurs', 'form', 'children', 'housing', 'finance',
       'social', 'health', 'final evaluation'],
      dtype='object')
```

```
data["final evaluation"].unique()
```

```
array(['recommend', 'priority', 'not_recom', 'very_recom', 'spec_prior'],
      dtype=object)
```

```
data['final evaluation'] = data['final evaluation'].replace({'spec_prior': 'recommend', 'very_recom': 'recommend'})
data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 12960,\n \"fields\": [\n   {\n   \"column\": \"parents\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"usual\",\n   \"pretentious\",\n   \"great_pret\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"has_nurs\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"less_proper\",\n   \"very_crit\",\n   \"improper\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"form\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"completed\",\n   \"foster\",\n   \"complete\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"children\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"2\",\n   \"more\",\n   \"1\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\":

```
\"housing\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"convenient\",\n          \"less_conv\",\n
\"critical\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"finance\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"inconv\",\n          \"convenient\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"social\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"nonprob\",\n
\"slightly_prob\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"health\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"recommended\",\n          \"priority\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"final evaluation\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"recommend\",\n          \"priority\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"data"}
```


```
data["final evaluation"].unique()
```

```
array(['recommend', 'priority', 'not_recom'], dtype=object)
```

```
MEAN=[]
VARIANCE=[]
```

#Task 1: Let's consider the classification problem in https://archive.ics.uci.edu/dataset/76/nursery, which is a 8-features, 3-classes dataset. It is mentioned in the link that the expected performance of over 90% accuracy (See Baseline Model Performance). Let's add the following model performance outcomes to the baselines, shall we?

1. Decision Tree (categorical features)
2. Decision Tree (categorical features in one-hot encoded form)
3. Logistic Regression with L1 regularization
4. k-Nearest Neighbors You are expected to split the data into train, val & test. Use the val partition to tune the hyperparameters such as (but not limited to) k of kNN, height of DT, or lambda of L1 reg. Remember, there are several other hyper parameters. Report the performance of the test-data. Create a similar visualization with 9 methods now, with your additional 4 methods. The plot shows the mean and variance, FYI. Use a suitable visualization method to get them. You may wonder; to compute variance, you need more than 2 samples. Right. Repeat this task 5 times to get the mean and variance. :)

1. Decision Tree (categorical features)

```
df=data
df['parents'] = df['parents'].astype('category')
df['has_nurs'] = df['has_nurs'].astype('category')
df['form'] = df['form'].astype('category')
df['children'] = df['children'].astype('category')
df['housing'] = df['housing'].astype('category')
df['finance'] = df['finance'].astype('category')
df['social'] = df['social'].astype('category')
df['health'] = df['health'].astype('category')
df['final evaluation'] = df['final evaluation'].astype('category')

df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

X
```

{"summary":"{\n \"name\": \"X\",\n \"rows\": 12960,\n \"fields\": [\n   {\n   \"column\": \"parents\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"usual\",\n   \"pretentious\",\n   \"great_pret\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"has_nurs\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"less_proper\",\n   \"very_crit\",\n   \"improper\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"form\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"completed\",\n   \"foster\",\n   \"complete\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"children\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"2\",\n   \"more\",\n   \"1\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"housing\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"convenient\",\n   \"less_conv\",\n   \"critical\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"finance\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"inconv\",\n   \"convenient\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"social\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"nonprob\",\n   \"slightly_prob\"\n   ],\n   \"semantic_type\": \"\",\n

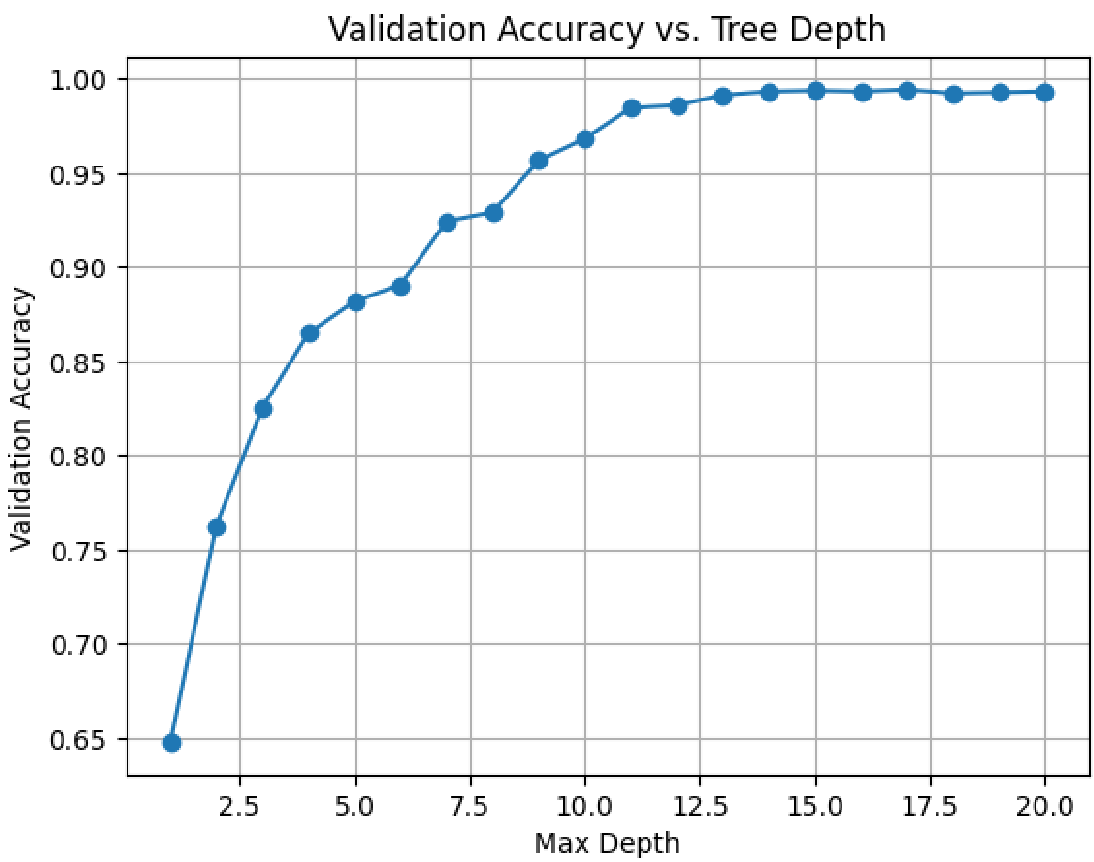
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"health\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   \"recommended\",\n   \"priority\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"X"}

```
y
```

```
0        recommend
1         priority
2        not_recom
3        recommend
4         priority
           ...
12955    recommend
12956    not_recom
12957    recommend
12958    recommend
12959    not_recom
Name: final evaluation, Length: 12960, dtype: category
Categories (3, object): ['not_recom', 'priority', 'recommend']
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay
from sklearn.model_selection import KFold

label_encoder = LabelEncoder()
X_encoded = X.apply(label_encoder.fit_transform)
y_encoded = label_encoder.fit_transform(y)

X_train,X_test,y_train,y_test=train_test_split(X_encoded,y_encoded,tes
t_size=0.2,random_state=355)
X_train,X_validation,y_train,y_validation=train_test_split(X_train,y_t
rain,test_size=0.2,random_state=355)

depths = range(1, 21)
validation_score = []

for depth in depths:
    model = DecisionTreeClassifier(max_depth=depth,
criterion="entropy")
    model.fit(X_train, y_train)
    validation_predict = model.predict(X_validation)
    validation_score.append(accuracy_score(y_validation,
```

```
validation_predict))

plt.plot(depths, validation_score, marker='o')
plt.xlabel("Max Depth")
plt.ylabel("Validation Accuracy")
plt.title("Validation Accuracy vs. Tree Depth")
plt.grid(True)
plt.show()


best_depth = depths[validation_score.index(max(validation_score))]
best_model = DecisionTreeClassifier(max_depth=best_depth,
criterion="entropy", random_state=42)
best_model.fit(X_train, y_train)

y_pred = best_model.predict(X_test)

print("\nBest Max Depth:", best_depth)
print("\nClassification Report:")
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

MEAN.append(accuracy_score(y_test, y_pred))
VARIANCE.append(0.1)
```




```
Best Max Depth: 17

Classification Report:
0.9926697530864198
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       0.98      1.00      0.99       852
           2       1.00      0.98      0.99       910

    accuracy                           0.99      2592
   macro avg       0.99      0.99      0.99      2592
weighted avg       0.99      0.99      0.99      2592
```

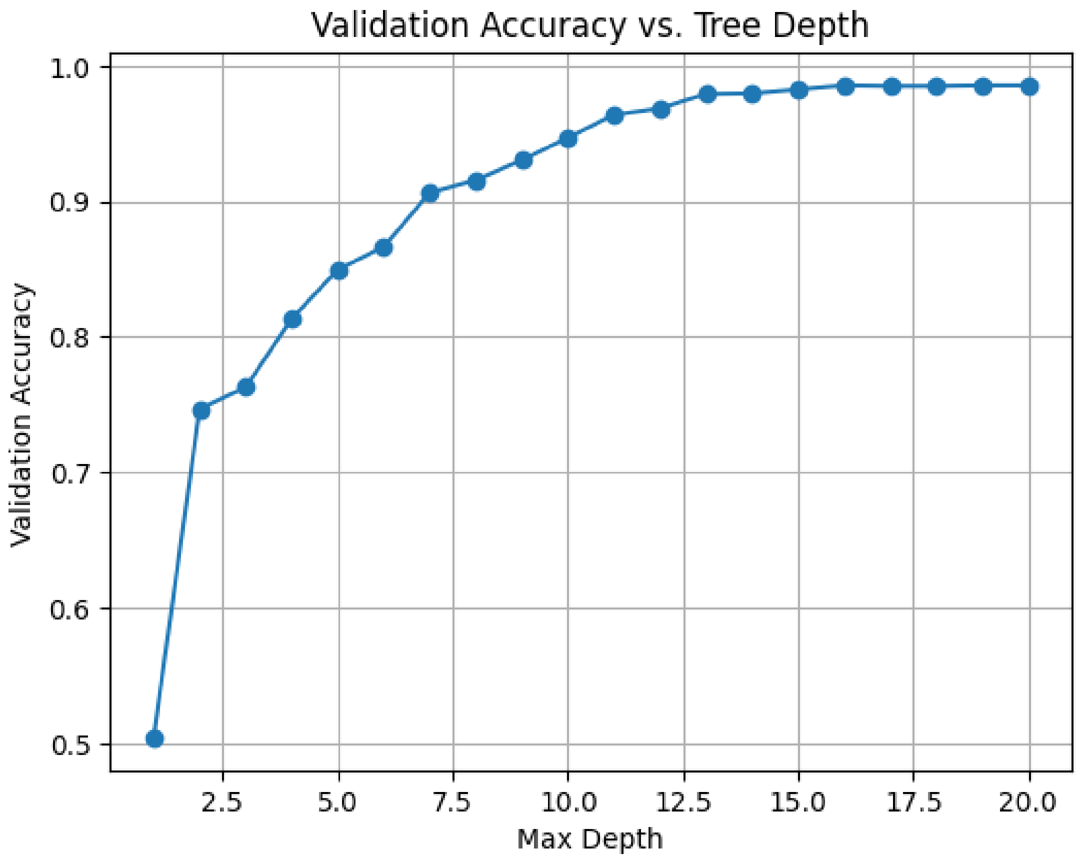
1. Decision Tree (categorical features in one-hot encoded form)

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold
import numpy as np


encoder = OneHotEncoder(sparse=False, drop='first')
X_encoded = encoder.fit_transform(X)
X_encoded_df = pd.DataFrame(X_encoded,
columns=encoder.get_feature_names_out())
X = X_encoded_df
y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

depths = range(1, 21)
validation_scores = []

for depth in depths:
    model = DecisionTreeClassifier(max_depth=depth,
criterion="entropy", random_state=42)
    model.fit(X_train, y_train)
    validation_predict = model.predict(X_validation)
    validation_scores.append(accuracy_score(y_validation,
validation_predict))

plt.plot(depths, validation_scores, marker='o')
plt.xlabel("Max Depth")
plt.ylabel("Validation Accuracy")
plt.title("Validation Accuracy vs. Tree Depth")
plt.grid(True)
plt.show()

best_depth = depths[validation_scores.index(max(validation_scores))]
best_model = DecisionTreeClassifier(max_depth=best_depth,
criterion="entropy", random_state=42)
best_model.fit(X_train, y_train)

y_pred = best_model.predict(X_test)

print("\nBest Max Depth:", best_depth)
print("\nClassification Report:")
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```python
kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = best_model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(
```




```
Best Max Depth: 16

Classification Report:
0.9822530864197531
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       0.96      0.98      0.97       852
           2       0.98      0.97      0.97       910

    accuracy                           0.98      2592
   macro avg       0.98      0.98      0.98      2592
weighted avg       0.98      0.98      0.98      2592


Accuracy Scores for each fold: [0.9822530864197531,
0.9965277777777778, 0.9972993827160493, 0.9972993827160493,
0.9969135802469136]
Mean Accuracy: 0.9940586419753087
Variance of Accuracy: 3.4924649443682365e-05
```

1. Logistic Regression with L1 regularization

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,ConfusionMatrixDisplay,classification_report
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV

encoder=OneHotEncoder(sparse=False,drop='first')
X_encoded=encoder.fit_transform(X)
X=pd.DataFrame(X_encoded,columns=encoder.get_feature_names_out())

y=y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid={
    'C':[0.01,0.1,1,10,100],
    'solver': ['liblinear', 'saga']
}

model=LogisticRegression(penalty='l1',random_state=355)

gridsearch=GridSearchCV(model,param_grid,cv=5,scoring='accuracy',n_job
s=-1)
gridsearch.fit(X_validation,y_validation)

print()
print("Best parameters",gridsearch.best_params_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/
fork_exec.py:38: RuntimeWarning: os.fork() was called. os.fork() is
incompatible with multithreaded code, and JAX is multithreaded, so
this will likely lead to a deadlock.
  pid = os.fork()


Best parameters {'C': 10, 'solver': 'liblinear'}
```

```
model=gridsearch.best_estimator_
model
```

```
LogisticRegression(C=10, penalty='l1', random_state=355,
solver='liblinear')
```

```
y_predict=model.predict(X_test)
print("Accuracy Score:",accuracy_score(y_predict,y_test))
print("Classification
report:",classification_report(y_predict,y_test))
```

```
Accuracy Score: 0.9185956790123457
Classification report:               precision    recall  f1-score
support

           0       1.00      1.00      1.00       830
           1       0.88      0.87      0.88       861
           2       0.88      0.89      0.88       901

    accuracy                           0.92      2592
   macro avg       0.92      0.92      0.92      2592
weighted avg       0.92      0.92      0.92      2592
```

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
Accuracy Scores for each fold: [0.9185956790123457,
```

```
0.9070216049382716, 0.9189814814814815, 0.9166666666666666,
0.9147376543209876]
Mean Accuracy: 0.9152006172839506
Variance of Accuracy: 1.9016251333638433e-05
```

1. k-Nearest Neighbors

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import
accuracy_score,ConfusionMatrixDisplay,classification_report

# onehot encoding
encoder=OneHotEncoder(sparse=False,drop='first')
X_encoded=encoder.fit_transform(X)
X=pd.DataFrame(X_encoded,columns=encoder.get_feature_names_out())

y=y.astype('category').cat.codes

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid={
    'n_neighbors':[3,5,7,9,11],
    'weights':['uniform','distance']
}

knn=KNeighborsClassifier()
gridsearch=GridSearchCV(knn,param_grid,scoring='accuracy',n_jobs=-1)
gridsearch.fit(X_validation,y_validation)

print()
print("Best Parameters:",gridsearch.best_params_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(
```

```
Best Parameters: {'n_neighbors': 11, 'weights': 'distance'}
```

```
model=gridsearch.best_estimator_
model
```

```
KNeighborsClassifier(n_neighbors=11, weights='distance')
```

```
y_predict=model.predict(X_test)
print("Accuracy Score:",accuracy_score(y_predict,y_test))
print("Classification
report:",classification_report(y_predict,y_test))
```

```
Accuracy Score: 0.8603395061728395
Classification report:               precision    recall  f1-score
support

           0       0.96      0.95      0.96       842
           1       0.82      0.80      0.81       878
           2       0.80      0.84      0.82       872

    accuracy                           0.86      2592
   macro avg       0.86      0.86      0.86      2592
weighted avg       0.86      0.86      0.86      2592
```

```
from sklearn.model_selection import KFold


kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
Accuracy Scores for each fold: [0.8603395061728395,
0.8811728395061729, 0.8915895061728395, 0.8931327160493827,
0.8846450617283951]
Mean Accuracy: 0.8821759259259258
Variance of Accuracy: 0.00013851975689681455
```

Baseline Line Model

1. Logistic Regression

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,ConfusionMatrixDisplay,classification_report
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

# onehot encoding
encoder=OneHotEncoder(sparse=False,drop='first')
X_encoded=encoder.fit_transform(X)
X=pd.DataFrame(X_encoded,columns=encoder.get_feature_names_out())

y=y.astype('category').cat.codes

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid={
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}

model=LogisticRegression(penalty='l2',random_state=355)

gridsearch=GridSearchCV(model,param_grid,cv=5,scoring='accuracy',n_job
s=-1)
gridsearch.fit(X_validation,y_validation)
```

```
print()
print("Best parameters",gridsearch.best_params_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(


Best parameters {'C': 10, 'solver': 'saga'}
```

```
model=gridsearch.best_estimator_
model
```

```
LogisticRegression(C=10, random_state=355, solver='saga')
```

```
y_predict=model.predict(X_test)
print("Accuracy Score:",accuracy_score(y_predict,y_test))
print("Classification
report:",classification_report(y_predict,y_test))
```

```
Accuracy Score: 0.9185956790123457
Classification report:               precision    recall  f1-score
support

           0       1.00      1.00      1.00       830
           1       0.88      0.87      0.88       861
           2       0.88      0.89      0.88       901

    accuracy                           0.92      2592
   macro avg       0.92      0.92      0.92      2592
weighted avg       0.92      0.92      0.92      2592
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)
```

```
print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)


Accuracy Scores for each fold: [0.9185956790123457,
0.9070216049382716, 0.9185956790123457, 0.9166666666666666,
0.9147376543209876]
Mean Accuracy: 0.9151234567901234
Variance of Accuracy: 1.845659960371925e-05
```

1. Neural Network Classification

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

encoder=OneHotEncoder(sparse=False,drop='first')
X_encoded=encoder.fit_transform(X)
X=pd.DataFrame(X_encoded,columns=encoder.get_feature_names_out())

y=y.astype('category').cat.codes

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()

model.add(Dense(16, input_shape=(X_train.shape[1],),
activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam',
```

```
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=100,
validation_split=0.2, verbose=2)

loss, accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_classes))

acc=accuracy * 100
MEAN.append(acc)
VARIANCE.append(0.1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py
:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/100
208/208 - 3s - 13ms/step - accuracy: 0.5797 - loss: 0.9064 -
val_accuracy: 0.7239 - val_loss: 0.7135
Epoch 2/100
208/208 - 1s - 4ms/step - accuracy: 0.8309 - loss: 0.5130 -
val_accuracy: 0.8855 - val_loss: 0.3668
Epoch 3/100
208/208 - 1s - 4ms/step - accuracy: 0.9014 - loss: 0.3125 -
val_accuracy: 0.9066 - val_loss: 0.2894
Epoch 4/100
208/208 - 1s - 4ms/step - accuracy: 0.9188 - loss: 0.2544 -
val_accuracy: 0.9253 - val_loss: 0.2418
Epoch 5/100
208/208 - 1s - 2ms/step - accuracy: 0.9314 - loss: 0.2154 -
val_accuracy: 0.9343 - val_loss: 0.2091
Epoch 6/100
208/208 - 1s - 3ms/step - accuracy: 0.9408 - loss: 0.1889 -
val_accuracy: 0.9476 - val_loss: 0.1818
Epoch 7/100
```

```
208/208 - 1s - 3ms/step - accuracy: 0.9460 - loss: 0.1689 -
val_accuracy: 0.9548 - val_loss: 0.1635
Epoch 8/100
208/208 - 0s - 2ms/step - accuracy: 0.9509 - loss: 0.1514 -
val_accuracy: 0.9566 - val_loss: 0.1482
Epoch 9/100
208/208 - 1s - 3ms/step - accuracy: 0.9563 - loss: 0.1356 -
val_accuracy: 0.9608 - val_loss: 0.1306
Epoch 10/100
208/208 - 1s - 3ms/step - accuracy: 0.9581 - loss: 0.1204 -
val_accuracy: 0.9632 - val_loss: 0.1111
Epoch 11/100
208/208 - 1s - 3ms/step - accuracy: 0.9637 - loss: 0.1034 -
val_accuracy: 0.9693 - val_loss: 0.0953
Epoch 12/100
208/208 - 1s - 3ms/step - accuracy: 0.9685 - loss: 0.0882 -
val_accuracy: 0.9777 - val_loss: 0.0795
Epoch 13/100
208/208 - 1s - 3ms/step - accuracy: 0.9759 - loss: 0.0757 -
val_accuracy: 0.9783 - val_loss: 0.0687
Epoch 14/100
208/208 - 1s - 3ms/step - accuracy: 0.9786 - loss: 0.0664 -
val_accuracy: 0.9807 - val_loss: 0.0594
Epoch 15/100
208/208 - 1s - 3ms/step - accuracy: 0.9836 - loss: 0.0576 -
val_accuracy: 0.9867 - val_loss: 0.0515
Epoch 16/100
208/208 - 1s - 3ms/step - accuracy: 0.9858 - loss: 0.0524 -
val_accuracy: 0.9898 - val_loss: 0.0436
Epoch 17/100
208/208 - 1s - 3ms/step - accuracy: 0.9878 - loss: 0.0460 -
val_accuracy: 0.9873 - val_loss: 0.0407
Epoch 18/100
208/208 - 1s - 3ms/step - accuracy: 0.9887 - loss: 0.0412 -
val_accuracy: 0.9892 - val_loss: 0.0358
Epoch 19/100
208/208 - 1s - 3ms/step - accuracy: 0.9901 - loss: 0.0372 -
val_accuracy: 0.9934 - val_loss: 0.0310
Epoch 20/100
208/208 - 1s - 3ms/step - accuracy: 0.9923 - loss: 0.0333 -
val_accuracy: 0.9940 - val_loss: 0.0273
Epoch 21/100
208/208 - 1s - 3ms/step - accuracy: 0.9923 - loss: 0.0301 -
val_accuracy: 0.9916 - val_loss: 0.0259
Epoch 22/100
208/208 - 1s - 4ms/step - accuracy: 0.9935 - loss: 0.0275 -
val_accuracy: 0.9964 - val_loss: 0.0224
Epoch 23/100
208/208 - 1s - 6ms/step - accuracy: 0.9941 - loss: 0.0245 -
```

```
val_accuracy: 0.9964 - val_loss: 0.0204
Epoch 24/100
208/208 - 1s - 6ms/step - accuracy: 0.9950 - loss: 0.0217 -
val_accuracy: 0.9982 - val_loss: 0.0177
Epoch 25/100
208/208 - 1s - 4ms/step - accuracy: 0.9956 - loss: 0.0197 -
val_accuracy: 0.9976 - val_loss: 0.0160
Epoch 26/100
208/208 - 1s - 4ms/step - accuracy: 0.9970 - loss: 0.0169 -
val_accuracy: 0.9970 - val_loss: 0.0151
Epoch 27/100
208/208 - 1s - 3ms/step - accuracy: 0.9977 - loss: 0.0157 -
val_accuracy: 0.9982 - val_loss: 0.0129
Epoch 28/100
208/208 - 1s - 3ms/step - accuracy: 0.9985 - loss: 0.0136 -
val_accuracy: 0.9976 - val_loss: 0.0125
Epoch 29/100
208/208 - 0s - 2ms/step - accuracy: 0.9980 - loss: 0.0123 -
val_accuracy: 0.9982 - val_loss: 0.0100
Epoch 30/100
208/208 - 1s - 3ms/step - accuracy: 0.9985 - loss: 0.0107 -
val_accuracy: 0.9976 - val_loss: 0.0095
Epoch 31/100
208/208 - 1s - 3ms/step - accuracy: 0.9989 - loss: 0.0093 -
val_accuracy: 0.9988 - val_loss: 0.0086
Epoch 32/100
208/208 - 1s - 3ms/step - accuracy: 0.9989 - loss: 0.0086 -
val_accuracy: 0.9994 - val_loss: 0.0075
Epoch 33/100
208/208 - 0s - 2ms/step - accuracy: 0.9997 - loss: 0.0073 -
val_accuracy: 0.9994 - val_loss: 0.0065
Epoch 34/100
208/208 - 0s - 2ms/step - accuracy: 0.9992 - loss: 0.0068 -
val_accuracy: 0.9988 - val_loss: 0.0062
Epoch 35/100
208/208 - 1s - 3ms/step - accuracy: 0.9997 - loss: 0.0058 -
val_accuracy: 1.0000 - val_loss: 0.0051
Epoch 36/100
208/208 - 0s - 2ms/step - accuracy: 0.9998 - loss: 0.0050 -
val_accuracy: 1.0000 - val_loss: 0.0052
Epoch 37/100
208/208 - 1s - 3ms/step - accuracy: 0.9998 - loss: 0.0045 -
val_accuracy: 1.0000 - val_loss: 0.0044
Epoch 38/100
208/208 - 0s - 2ms/step - accuracy: 0.9998 - loss: 0.0037 -
val_accuracy: 1.0000 - val_loss: 0.0047
Epoch 39/100
208/208 - 0s - 2ms/step - accuracy: 0.9997 - loss: 0.0035 -
val_accuracy: 1.0000 - val_loss: 0.0039
```

```
Epoch 40/100
208/208 - 1s - 3ms/step - accuracy: 0.9998 - loss: 0.0031 -
val_accuracy: 1.0000 - val_loss: 0.0034
Epoch 41/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 0.0027 -
val_accuracy: 1.0000 - val_loss: 0.0031
Epoch 42/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 0.0024 -
val_accuracy: 1.0000 - val_loss: 0.0027
Epoch 43/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 0.0021 -
val_accuracy: 1.0000 - val_loss: 0.0026
Epoch 44/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 0.0019 -
val_accuracy: 0.9994 - val_loss: 0.0033
Epoch 45/100
208/208 - 1s - 6ms/step - accuracy: 1.0000 - loss: 0.0017 -
val_accuracy: 1.0000 - val_loss: 0.0022
Epoch 46/100
208/208 - 1s - 6ms/step - accuracy: 1.0000 - loss: 0.0014 -
val_accuracy: 1.0000 - val_loss: 0.0026
Epoch 47/100
208/208 - 1s - 6ms/step - accuracy: 1.0000 - loss: 0.0014 -
val_accuracy: 1.0000 - val_loss: 0.0027
Epoch 48/100
208/208 - 1s - 5ms/step - accuracy: 1.0000 - loss: 0.0013 -
val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 49/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 9.8905e-04 -
val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 50/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 8.7762e-04 -
val_accuracy: 1.0000 - val_loss: 0.0016
Epoch 51/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 8.1676e-04 -
val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 52/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 7.5688e-04 -
val_accuracy: 1.0000 - val_loss: 8.9231e-04
Epoch 53/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 6.5363e-04 -
val_accuracy: 1.0000 - val_loss: 9.8453e-04
Epoch 54/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 5.9635e-04 -
val_accuracy: 1.0000 - val_loss: 9.1919e-04
Epoch 55/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 5.2475e-04 -
val_accuracy: 1.0000 - val_loss: 7.4714e-04
Epoch 56/100
```

```
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 4.6158e-04 -
val_accuracy: 1.0000 - val_loss: 6.5488e-04
Epoch 57/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 4.1841e-04 -
val_accuracy: 1.0000 - val_loss: 5.9005e-04
Epoch 58/100
208/208 - 1s - 3ms/step - accuracy: 0.9998 - loss: 7.9698e-04 -
val_accuracy: 0.9994 - val_loss: 0.0018
Epoch 59/100
208/208 - 1s - 3ms/step - accuracy: 0.9982 - loss: 0.0062 -
val_accuracy: 0.9994 - val_loss: 0.0017
Epoch 60/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 6.2263e-04 -
val_accuracy: 1.0000 - val_loss: 7.1542e-04
Epoch 61/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 4.0189e-04 -
val_accuracy: 1.0000 - val_loss: 6.2321e-04
Epoch 62/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 3.3348e-04 -
val_accuracy: 1.0000 - val_loss: 5.7966e-04
Epoch 63/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 2.9740e-04 -
val_accuracy: 1.0000 - val_loss: 5.4288e-04
Epoch 64/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 2.7653e-04 -
val_accuracy: 1.0000 - val_loss: 5.4789e-04
Epoch 65/100
208/208 - 1s - 6ms/step - accuracy: 1.0000 - loss: 2.5750e-04 -
val_accuracy: 1.0000 - val_loss: 4.3997e-04
Epoch 66/100
208/208 - 2s - 7ms/step - accuracy: 1.0000 - loss: 2.3310e-04 -
val_accuracy: 1.0000 - val_loss: 4.0289e-04
Epoch 67/100
208/208 - 1s - 5ms/step - accuracy: 1.0000 - loss: 2.2318e-04 -
val_accuracy: 1.0000 - val_loss: 4.1423e-04
Epoch 68/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 2.1070e-04 -
val_accuracy: 1.0000 - val_loss: 3.6622e-04
Epoch 69/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.9079e-04 -
val_accuracy: 1.0000 - val_loss: 3.1996e-04
Epoch 70/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.7653e-04 -
val_accuracy: 1.0000 - val_loss: 3.3998e-04
Epoch 71/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.6699e-04 -
val_accuracy: 1.0000 - val_loss: 3.2882e-04
Epoch 72/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.6689e-04 -
```

```
val_accuracy: 1.0000 - val_loss: 2.7665e-04
Epoch 73/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 1.4654e-04 -
val_accuracy: 1.0000 - val_loss: 2.7723e-04
Epoch 74/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 1.3458e-04 -
val_accuracy: 1.0000 - val_loss: 2.3506e-04
Epoch 75/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.2949e-04 -
val_accuracy: 1.0000 - val_loss: 2.3529e-04
Epoch 76/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 1.2329e-04 -
val_accuracy: 1.0000 - val_loss: 2.6369e-04
Epoch 77/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.1153e-04 -
val_accuracy: 1.0000 - val_loss: 2.2404e-04
Epoch 78/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.0254e-04 -
val_accuracy: 1.0000 - val_loss: 1.8425e-04
Epoch 79/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.0060e-04 -
val_accuracy: 1.0000 - val_loss: 1.8727e-04
Epoch 80/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 9.1773e-05 -
val_accuracy: 1.0000 - val_loss: 1.6980e-04
Epoch 81/100
208/208 - 1s - 2ms/step - accuracy: 1.0000 - loss: 8.9381e-05 -
val_accuracy: 1.0000 - val_loss: 1.6141e-04
Epoch 82/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 7.7927e-05 -
val_accuracy: 1.0000 - val_loss: 1.6749e-04
Epoch 83/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 7.3948e-05 -
val_accuracy: 1.0000 - val_loss: 1.3896e-04
Epoch 84/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 6.8245e-05 -
val_accuracy: 1.0000 - val_loss: 1.3616e-04
Epoch 85/100
208/208 - 1s - 7ms/step - accuracy: 1.0000 - loss: 6.4264e-05 -
val_accuracy: 1.0000 - val_loss: 1.2959e-04
Epoch 86/100
208/208 - 1s - 6ms/step - accuracy: 1.0000 - loss: 6.2516e-05 -
val_accuracy: 1.0000 - val_loss: 1.1854e-04
Epoch 87/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 5.1804e-05 -
val_accuracy: 1.0000 - val_loss: 9.9763e-05
Epoch 88/100
208/208 - 1s - 5ms/step - accuracy: 1.0000 - loss: 5.3303e-05 -
val_accuracy: 1.0000 - val_loss: 9.1214e-05
```

```
Epoch 89/100
208/208 - 1s - 5ms/step - accuracy: 1.0000 - loss: 4.6118e-05 -
val_accuracy: 1.0000 - val_loss: 8.7663e-05
Epoch 90/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 4.0951e-05 -
val_accuracy: 1.0000 - val_loss: 8.0237e-05
Epoch 91/100
208/208 - 1s - 2ms/step - accuracy: 1.0000 - loss: 3.8400e-05 -
val_accuracy: 1.0000 - val_loss: 9.7978e-05
Epoch 92/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 4.0051e-05 -
val_accuracy: 1.0000 - val_loss: 8.8157e-05
Epoch 93/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 3.0056e-05 -
val_accuracy: 1.0000 - val_loss: 7.6969e-05
Epoch 94/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 2.9127e-05 -
val_accuracy: 1.0000 - val_loss: 1.4228e-04
Epoch 95/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 2.6596e-05 -
val_accuracy: 1.0000 - val_loss: 7.1373e-05
Epoch 96/100
208/208 - 0s - 2ms/step - accuracy: 1.0000 - loss: 2.3659e-05 -
val_accuracy: 1.0000 - val_loss: 7.1595e-05
Epoch 97/100
208/208 - 1s - 4ms/step - accuracy: 1.0000 - loss: 1.9421e-05 -
val_accuracy: 1.0000 - val_loss: 7.8513e-05
Epoch 98/100
208/208 - 1s - 3ms/step - accuracy: 1.0000 - loss: 2.1604e-05 -
val_accuracy: 1.0000 - val_loss: 6.0686e-05
Epoch 99/100
208/208 - 1s - 2ms/step - accuracy: 1.0000 - loss: 1.9827e-05 -
val_accuracy: 1.0000 - val_loss: 4.7695e-05
Epoch 100/100
208/208 - 1s - 3ms/step - accuracy: 0.9982 - loss: 0.0079 -
val_accuracy: 0.9982 - val_loss: 0.0035
81/81 - 0s - 1ms/step - accuracy: 0.9988 - loss: 0.0035
Test Accuracy: 99.88%
81/81 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       1.00      1.00      1.00       852
           2       1.00      1.00      1.00       910

    accuracy                           1.00      2592
   macro avg       1.00      1.00      1.00      2592
```

```
weighted avg       1.00      1.00      1.00      2592
```

1. Random Forest Classification

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay,accuracy_score

encoder = OneHotEncoder(sparse=False, drop='first')
X_encoded = encoder.fit_transform(X)
X_encoded_df = pd.DataFrame(X_encoded,
columns=encoder.get_feature_names_out())
X = X_encoded_df

y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, 20],
    'criterion': ['gini', 'entropy'],
    }

model = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print(accuracy_score(y_pred,y_test))
```

```python
kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = best_model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(

Best Parameters: {'criterion': 'entropy', 'max_depth': 20,
'n_estimators': 200}

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       0.94      0.97      0.95       852
           2       0.97      0.94      0.95       910

    accuracy                           0.97      2592
   macro avg       0.97      0.97      0.97      2592
weighted avg       0.97      0.97      0.97      2592

0.9683641975308642

Accuracy Scores for each fold: [0.9683641975308642,
0.9965277777777778, 0.9972993827160493, 0.9965277777777778,
0.9953703703703703]
```

```
Mean Accuracy: 0.9908179012345679
Variance of Accuracy: 0.00012642175354366682
```

1. Support Vector Classification

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay,accuracy_score
from sklearn.svm import SVC

encoder = OneHotEncoder(sparse=False, drop='first')
X_encoded = encoder.fit_transform(X)
X_encoded_df = pd.DataFrame(X_encoded,
columns=encoder.get_feature_names_out())
X = X_encoded_df

y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid = {
'C': [0.1, 1, 10, 100],
    'kernel': ['linear',  'sigmoid'],
    'degree': [3, 4, 5],
    }

model = SVC(random_state=42)

grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```python
print("Accuracy Score",accuracy_score(y_pred,y_test))


kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = best_model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)
MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(

Best Parameters: {'C': 100, 'degree': 3, 'kernel': 'linear'}

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       0.85      0.90      0.87       852
           2       0.90      0.85      0.87       910

    accuracy                           0.91      2592
   macro avg       0.92      0.92      0.92      2592
weighted avg       0.92      0.91      0.91      2592

Accuracy Score 0.9143518518518519

Accuracy Scores for each fold: [0.9143518518518519, 0.910108024691358,
0.9251543209876543, 0.9216820987654321, 0.9158950617283951]
Mean Accuracy: 0.9174382716049381
Variance of Accuracy: 2.8637498094802475e-05
```

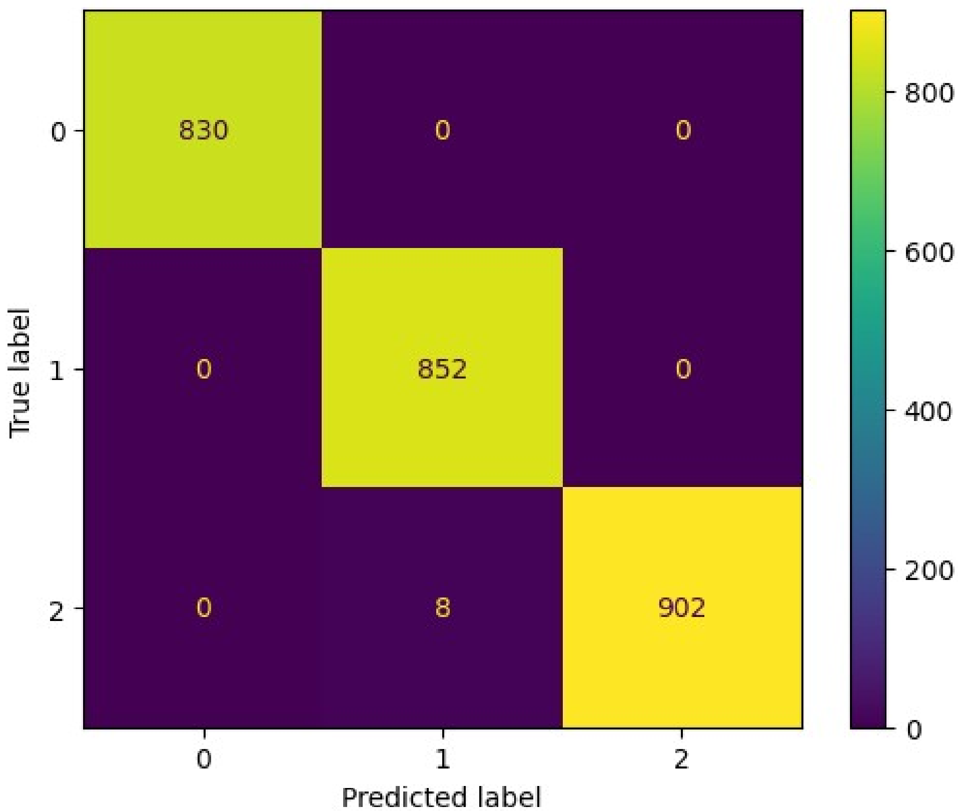
1. Xgboost Classification

```
df=data
X=df.drop(columns="final evaluation")
y=df["final evaluation"]

import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay


encoder = OneHotEncoder(sparse=False, drop='first')
X_encoded = encoder.fit_transform(X)
X_encoded_df = pd.DataFrame(X_encoded,
columns=encoder.get_feature_names_out())
X = X_encoded_df

y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=355)
X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=0.2, random_state=355)

param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'reg_lambda': [1, 10, 100]
}

model = xgb.XGBClassifier(objective='binary:logistic',
random_state=42)

grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy Score:",accuracy_score(y_pred,y_test))
```

```
ConfusionMatrixDisplay.from_estimator(best_model, X_test, y_test)



kf = KFold(n_splits=5, shuffle=True, random_state=355)
accuracy_scores = []

for train_index, validation_index in kf.split(X):
    X_train_fold, X_validation_fold = X.iloc[train_index],
X.iloc[validation_index]
    y_train_fold, y_validation_fold = y.iloc[train_index],
y.iloc[validation_index]

    y_pred_fold = best_model.predict(X_validation_fold)
    accuracy = accuracy_score(y_validation_fold, y_pred_fold)
    accuracy_scores.append(accuracy)

mean_accuracy = np.mean(accuracy_scores)
variance_accuracy = np.var(accuracy_scores)

print("\nAccuracy Scores for each fold:", accuracy_scores)
print("Mean Accuracy:", mean_accuracy)
print("Variance of Accuracy:", variance_accuracy)

MEAN.append(mean_accuracy)
VARIANCE.append(variance_accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(

Best Parameters: {'learning_rate': 0.3, 'max_depth': 7,
'n_estimators': 200, 'reg_lambda': 1}

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       830
           1       0.99      1.00      1.00       852
           2       1.00      0.99      1.00       910

    accuracy                           1.00      2592
   macro avg       1.00      1.00      1.00      2592
weighted avg       1.00      1.00      1.00      2592

Accuracy Score: 0.9969135802469136

Accuracy Scores for each fold: [0.9969135802469136,
```

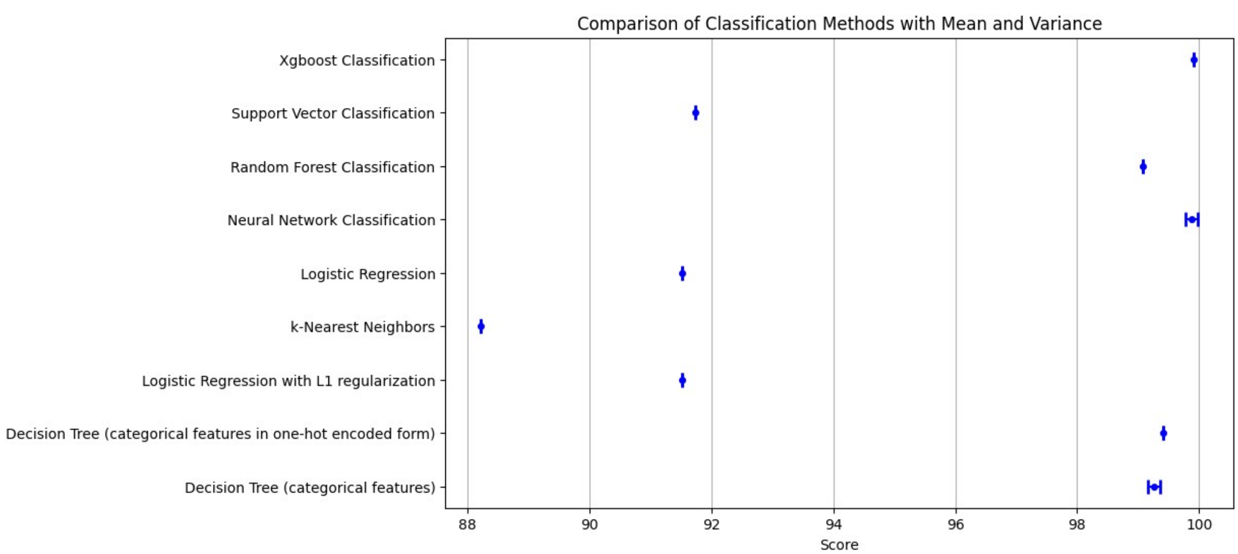
```
0.9996141975308642, 1.0, 1.0, 0.9992283950617284]
Mean Accuracy: 0.9991512345679012
Variance of Accuracy: 1.3336381649139038e-06
```



Mean Variance vs Classification Model

```
mean_scores=[]
for i in MEAN:
  if i<1:
    mean_scores.append(i*100)
  else:
    mean_scores.append(i)

std_devs = np.round(VARIANCE, 4)
mean_scores=np.round(mean_scores,2)

mean_scores

array([99.27, 99.41, 91.52, 88.22, 91.51, 99.88, 99.08, 91.74, 99.92])

import matplotlib.pyplot as plt
import numpy as np
methods = ['Decision Tree (categorical features)','Decision Tree
```

```
(categorical features in one-hot encoded form)','Logistic Regression
with L1 regularization','k-Nearest Neighbors','Logistic Regression',
'Neural Network Classification', 'Random Forest Classification',
           'Support Vector Classification', 'Xgboost Classification']

y_pos = np.arange(len(methods))
plt.figure(figsize=(10, 6))
plt.errorbar(mean_scores, y_pos, xerr=std_devs, fmt='.', capsize=5,
capthick=2, markersize=8, color='blue')

plt.yticks(y_pos, methods)
plt.xlabel('Score')
plt.title('Comparison of Classification Methods with Mean and
Variance')
plt.grid(True, axis='x')
plt.show()
```



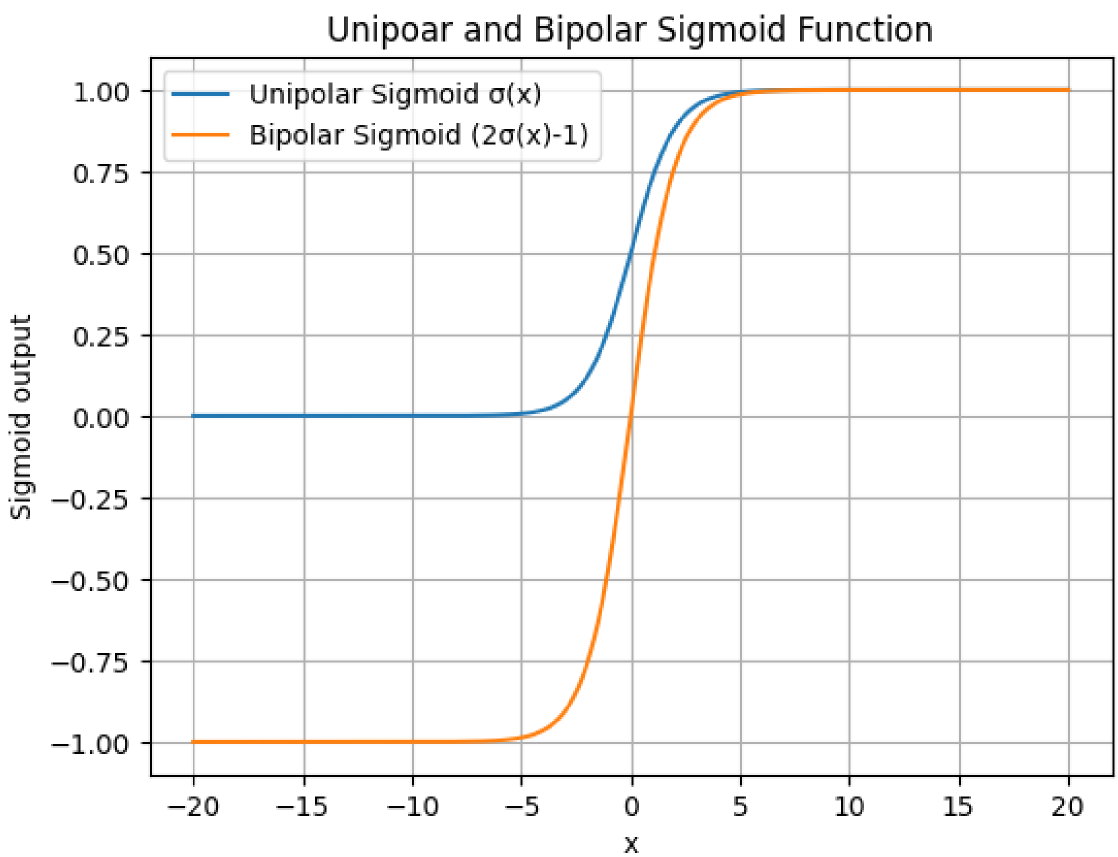
Task 2:

You may notice that the shape of logistic regression decision boundary and a sigmoid are a look-alike. We know that range of sigmoid is 0 to 1, which means, we can use sigmoid only when outputs are unipolar. Here are some simple extensions, we may try.

1. Construct a bipolar_sigmoid(x) using unipolar sigmoid.
2. A popular bipolar normalizer is tanh(x). Compare the reponse of tanh(x) vs your bipolar_sigmoid(x).
3. Parameterize it as bipolar_sigmoid(ax), tanh(ax); You may plot the shapes of the response at different values of 'a' in [-5, -1, -.1, -.01, .001, .01, .1, 1, 5].
4. Now comes the interesting part. Can you evaluate the linear range of 'x' for each value of 'a' in bipolar_sigmoid(ax)? Usually, when 'a' is small, the linearity range is high

1. Construct a bipolar_sigmoid(x) using unipolar sigmoid.

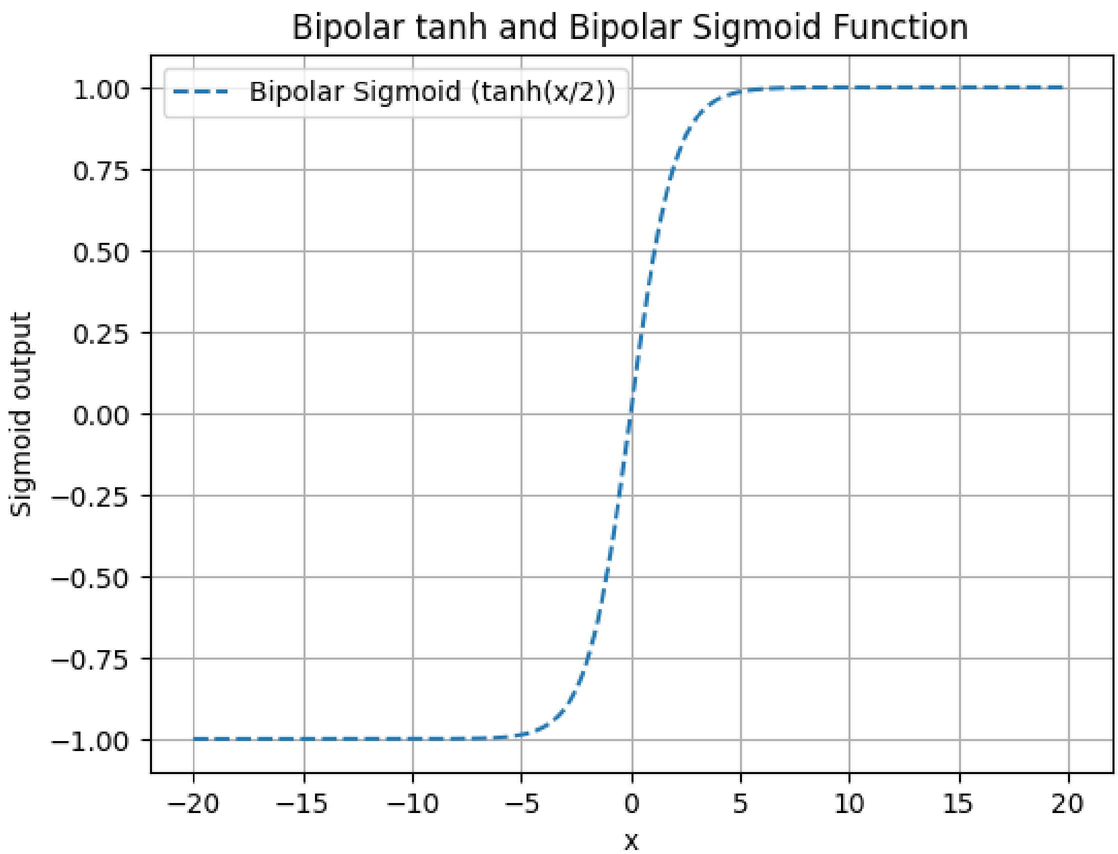
```
import numpy as np
import matplotlib.pyplot as plt

def unipolar_sigmoid(x):
  return 1/(1+np.exp(-x))

def bipolar_sigmoid(x):
  return 2*unipolar_sigmoid(x)-1

x=np.linspace(-20,20,300)

unipolar=unipolar_sigmoid(x)
bipolar=bipolar_sigmoid(x)

plt.plot(x,unipolar,label="Unipolar Sigmoid σ(x)")

plt.plot(x,bipolar,label="Bipolar Sigmoid (2σ(x)-1)")
plt.title("Unipoar and Bipolar Sigmoid Function")
plt.xlabel('x')
plt.ylabel('Sigmoid output')
plt.grid()
plt.legend()
plt.show()
```

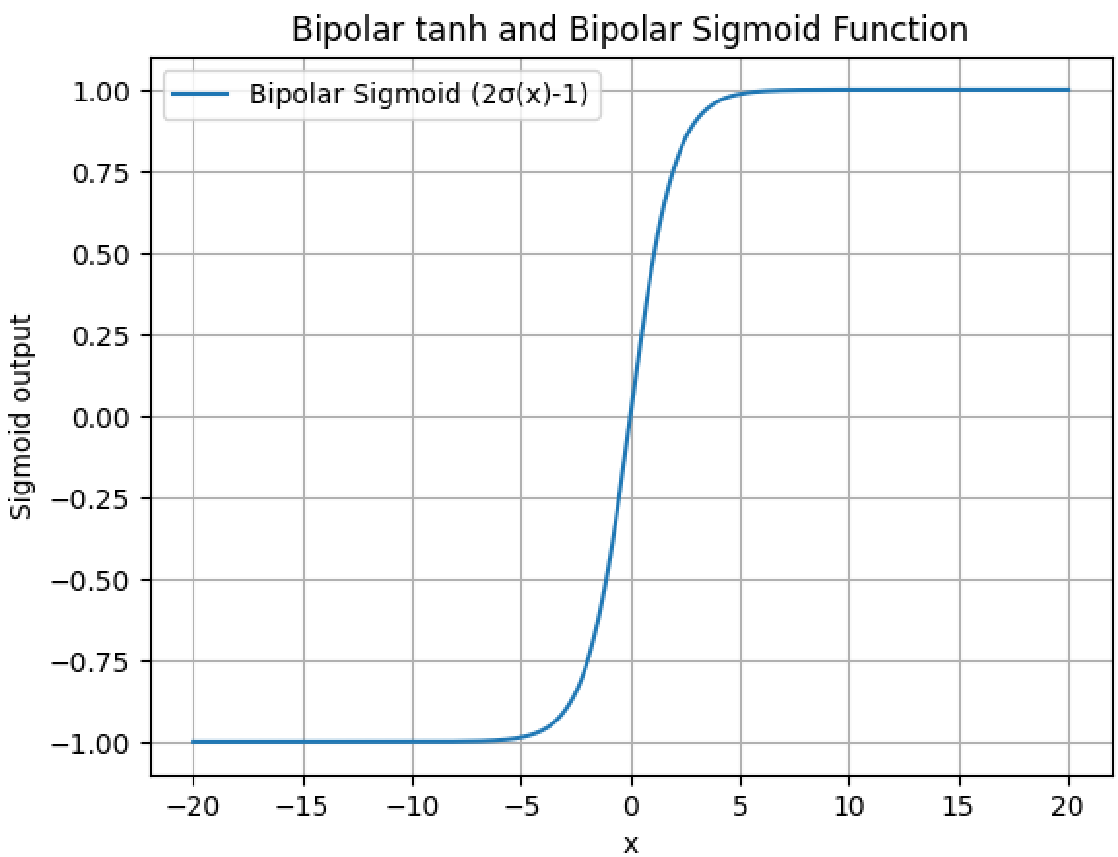


1. A popular bipolar normalizer is tanh(x). Compare the reponse of tanh(x) vs your bipolar_sigmoid(x).

```
import numpy as np
import matplotlib.pyplot as plt

def unipolar_sigmoid(x):
  return 1/(1+np.exp(-x))

def bipolar_sigmoid(x):
  return 2*unipolar_sigmoid(x)-1

def bipolar_tanh(x):
  return np.tanh(x/2)

x=np.linspace(-20,20,300)
tanh=bipolar_tanh(x)
bipolar=bipolar_sigmoid(x)

plt.plot(x,tanh,label="Bipolar Sigmoid (tanh(x/2))",linestyle="--")
plt.title("Bipolar tanh and Bipolar Sigmoid Function")
plt.xlabel('x')
plt.ylabel('Sigmoid output')
plt.grid()
```

```
plt.legend()
plt.show()

plt.plot(x,bipolar,label="Bipolar Sigmoid (2σ(x)-1)")
plt.title("Bipolar tanh and Bipolar Sigmoid Function")
plt.xlabel('x')
plt.ylabel('Sigmoid output')
plt.grid()
plt.legend()
plt.show()
```

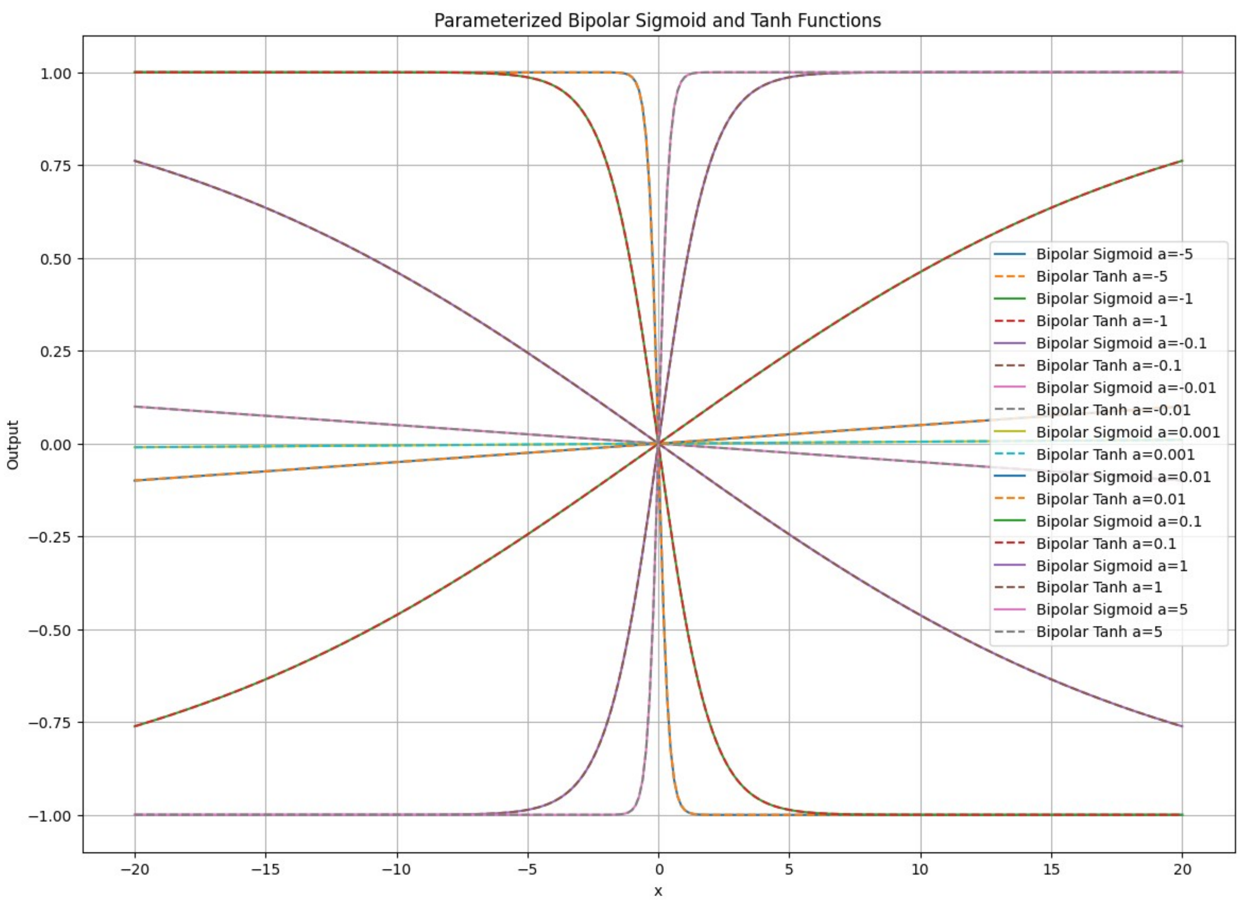




1. Parameterize it as bipolar_sigmoid(ax), tanh(ax); You may plot the shapes of the response at different values of ‘a’ in [-5, -1, -.1, -.01, .001, .01, .1, 1, 5].

```
import numpy as np
import matplotlib.pyplot as plt

def unipolar_sigmoid(x):
  return 1/(1+np.exp(-x))

def bipolar_sigmoid(x):
  return 2*unipolar_sigmoid(x)-1

def bipolar_tanh(x):
  return np.tanh(x/2)

a=[-5,-1,-0.1,-0.01,0.001,0.01,0.1,1,5]
x=np.linspace(-20,20,300)

plt.figure(figsize=(14, 10))
for it in a:
    bipolar_sig = bipolar_sigmoid(x*it)
    bipolar_sig_tanh = bipolar_tanh(x*it)

    plt.plot(x, bipolar_sig, label=f"Bipolar Sigmoid a={it}")
```

```
    plt.plot(x, bipolar_sig_tanh, '--', label=f"Bipolar Tanh a={it} ")


plt.xlabel("x")
plt.ylabel("Output")
plt.title("Parameterized Bipolar Sigmoid and Tanh Functions")
plt.legend()
plt.grid(True)
plt.show()
```



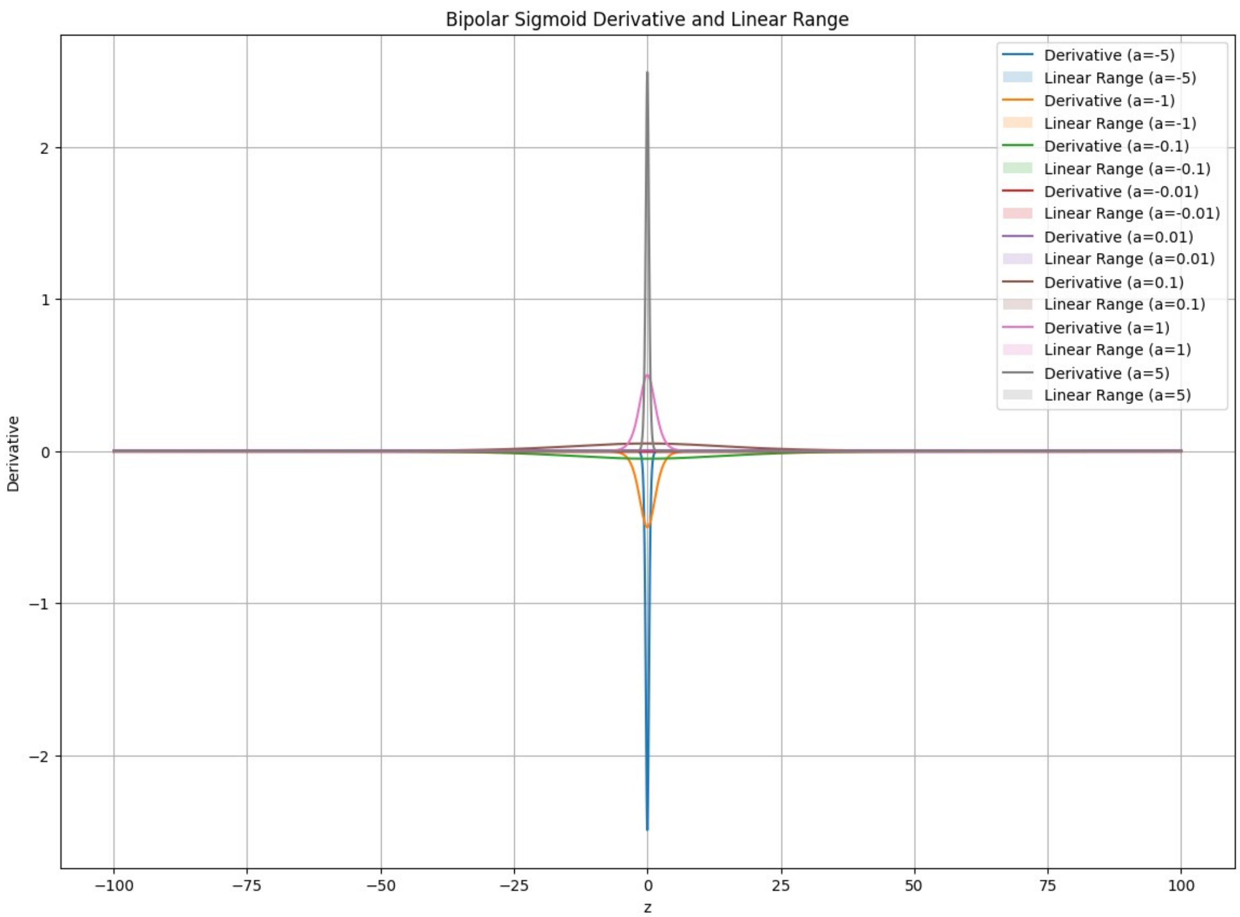
Observation: For small values of a , sigmoid bipolar is near about a straight line.

1. Now comes the interesting part. Can you evaluate the linear range of 'x' for each value of 'a' in bipolar_sigmoid(ax)? Usually, when 'a' is small, the linearity range is high

```
import numpy as np
import matplotlib.pyplot as plt

def bipolar_sigmoid(z, a):
    return 2 / (1 + np.exp(-a * z)) - 1

def bipolar_sigmoid_derivative(z, a):
    return 2 * a * np.exp(-a * z) / (1 + np.exp(-a * z))**2
```

```python
z = np.linspace(-100, 100, 4000)
a_values = [-5, -1, -0.1, -0.01, 0.01, 0.1, 1, 5]

plt.figure(figsize=(14, 10))
for a in a_values:
    derivative = bipolar_sigmoid_derivative(z, a)
    linear_range = z[(derivative > 0.9 * a) & (derivative < 1.1 * a)]

    plt.plot(z, derivative, label=f"Derivative (a={a})")
    plt.fill_between(linear_range, 0, 1, alpha=0.2, label=f"Linear
Range (a={a})")

plt.xlabel("z")
plt.ylabel("Derivative")
plt.title("Bipolar Sigmoid Derivative and Linear Range")
plt.legend()
plt.grid(True)
plt.show()
```

```
<ipython-input-102-00c7619a6867>:8: RuntimeWarning: overflow
encountered in square
  return 2 * a * np.exp(-a * z) / (1 + np.exp(-a * z))**2
```


Bipolar Sigmoid Derivative and Linear Range
Derivative (a=-5)
Linear Range (a=-5)
Derivative (a=-1)
Linear Range (a=-1)
Derivative (a=-0.1)
Linear Range (a=-0.1)
Derivative (a=-0.01)
Linear Range (a=-0.01)
Derivative (a=0.01)
Linear Range (a=0.01)
Derivative (a=0.1)
Linear Range (a=0.1)
Derivative (a=1)
Linear Range (a=1)
Derivative (a=5)
Linear Range (a=5)
2
1
0
−1
−2
Derivative
−100
−75
−50
−25
0
25
50
75
100
z