2025

SDSS CLASSIFICATION

This project utilizes a dataset from the Sloan Digital Sky Survey (SDSS) to develop a machine learning pipeline for both regression and classification tasks

Prepared by
VEERAMALLA BALAJI BHARGAV
STUDENT, IIIT KOTTAYAM

VEERAMALLA BALAJI BHARGAV
balaji.ijb@gmail.com
Student
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, KOTTAYAM

INDEX

PROJECT INITIALIZATION AND PLANNING PHASE

Define Problem Statement

Problem Statement	I am	I'm trying to	But	Because	What makes me feel
1. Galaxy morphology classification	Astronomer	Identify trends and patterns of galaxy morphology	It takes long time and manually not scallable	Of large datasets	frustrated
2. Galaxy Redshift Estimation	Astronomer	Map three dimensional distribution and investigate galaxies	It takes long time	To estimate red shifts from datasets	frustrated
3. Active Galactic Nuclei(AGN) Identification	Astronomer	Study properties of AGN	It takes long time	To identify an AGN candidate	frustrated

Project Proposal (Proposed Solution)

The proposal report aims to transform **Sloan Digital Sky Survey (SDSS)** galaxy classification using machine learning, boosting efficiency and accuracy. It tackles system inefficiencies, promising better operations, reduced risks, and happier customers. Key features include a machine learning-based credit model and real-time decision-making.

Project Overview

Objective	The primary objective is to revolutionize the **Sloan Digital Sky Survey (SDSS)** galaxy classification by implementing advanced machine learning techniques, ensuring faster and more accurate assessments.
Scope	The project comprehensively assesses and enhances the SDSS Classification, incorporating machine learning for a more robust and efficient system.

Problem Statement

Scenario – 1 Galaxy Morphology Classification

Description	Astronomers are interested in studying the morphology of galaxies to understand their formation and evolution processes. By utilizing machine learning techniques, researchers can train a classification model to categorize galaxies into different morphological types such as elliptical, spiral, or irregular.
Impact	This automated classification process enables astronomers to analyze large datasets of galaxy images efficiently and identify trends or patterns related to galaxy morphology.

Scenario – 2 Galaxy Redshift Estimation

Description	Redshift, which indicates the extent to which light from a galaxy has been shifted towards longer wavelengths due to the expansion of the universe, is a crucial parameter for studying cosmic distances and cosmological phenomena.
Impact	Machine learning models can be trained to estimate galaxy redshifts based on features extracted from their spectra or photometric properties measured by SDSS. Accurate redshift estimation enables astronomers to map the three-dimensional distribution of galaxies in the universe and investigate large-scale structures such as galaxy clusters and filaments.

Scenario – 3 Active Galactic Nuclei (AGN) Identification

Description	Galaxies hosting active galactic nuclei (AGN) exhibit intense emission from a compact region at their centers, powered by accretion onto supermassive black holes. Identifying AGN candidates from SDSS data is essential for studying their properties and understanding their impact on galaxy evolution.
Impact	Machine learning algorithms can be trained to recognize characteristic signatures of AGN in galaxy spectra or multi-wavelength photometric data, facilitating the automated identification of AGN hosts within large galaxy surveys like SDSS. This enables astronomers to conduct statistical analyses of AGN properties and investigate their role in galaxy formation and evolution processes.

Proposed Statement

Approach	Employing machine learning techniques to analyze and predict creditworthiness, creating a dynamic and adaptable loan approval system.
Key Features	- Implementation of a machine learning-based credit assessment model. - Real-time decision-making for quicker loan approvals. - Continuous learning to adapt to evolving financial landscapes.

Resource Requirements

Resource Type	**Description**	**Specification / Allocation**
Computing Resources	CPU/GPU specifications, number of cores	T4 GPU
Memory	RAM specifications	8 GB
Storage	Disk space for data, models, and logs	1 TB SSD

Software

Frameworks	Python frameworks	Flask
Libraries	Additional libraries	scikit-learn, pandas, numpy, matplotlib, seaborn
Development Environment	IDE	Jupyter Notebook, pycharm

Data

Data	Source, size, format	Kaggle dataset, 614, csv UCI dataset, 690, csv

The columns named “morphology” and “agni”(AGN Identification) in the dataset are not real values. They are add with random values

Morphology column

Value allocated – Shape

0 – Spiral

1– Elliptical

2– irregular

agni column

value – meaning

0 – AGN not identified

1 – AGN identified

Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (planned)
Sprint-1	Data Collection and Preprocessing	SL -3	Understanding and Loading data	Low	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-1	Data Collection and Preprocessing	SL -4	Data cleaning	High	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-1	Data Collection and Preprocessing	SL -5	EDA	Medium	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-4	Project Report	SL -20	Report	Medium	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-2	Model Development	SL -8	Training and model	Medium	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-2	Model Development	SL -9	Evaluating the model	Medium	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-2	Model Tuning and testing	SL -13	Model tuning	High	V. Balaji Bhargav	16/5/2025	29/5/2025
Sprint-	Model Tuning	SL -14	Model testing	Medium	V. Balaji	16/5/202	29/5/202

DATA COLLECTION AND PREPROCESSING

Data Collection Plan, Raw Data Sources Identified, Data Quality

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan:

Section	Description
Project Overview	The machine learning project aims to predict loan approval based on applicant information. Using a dataset with features in dataset, the objective is to build a model that accurately classifies AGN (approved or denied), morphology(spiral, elliptical, or irregular) and predict Redshift
Data Collection Plan	● Search for datasets related to Morphology, AGN and Redshift details. ● Prioritize datasets with diverse demographic information.
Raw Data Sources Identified	The raw data sources for this project include datasets obtained from Kaggle & UCI, the popular platforms for data science competitions and repositories. The provided sample data represents a subset of the collected information, encompassing variables such as Redshift, Morphology, and AGN.

Raw Data Sources Report:

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Dataset	The dataset comprises target variable(Predicting variable) only Redshift but not Morphology and AGN related columns so all the values in the that both columns are Random values.	https://www.kaggle.com/datasets/bryancimo/sdss-galaxy-classification-dr18	CSV	42000 KB	Public

Data Quality Report:

The Data Quality Report will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Quality Report:

Data Source	Data Quality Issue	Severity	Resolution Plan
Kaggle Dataset	Missing values in the u, g, r, i, z,petroR50_u, petroR50_g, petroR50_i, petroR50_r, petroR50_z, psfMag_u, psfMag_r, psfMag_g, psfMag_i, psfMag_z, expAB_u, expAB_g, expAB_r,	Moderate	Use mean/mode/median imputation

	expAB_i, expAB_z		
Kaggle Dataset	Categorial data in the dataset	Moderate	Encoding has to be done in the data

Data Exploration and Preprocessing Report

Dataset variables will be statistically analyzed to identify patterns and outliers, with Python employed for preprocessing tasks like normalization and feature engineering. Data cleaning will address missing values and outliers, ensuring quality for subsequent analysis and modeling, and forming a strong foundation for insights and predictions.

Section	**Description**
Data Overview	Dimensions: 100000 rows × 39 Columns

	u	g	r	i	z
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	19.621221	18.360874	17.723881	17.364947	17.153703
std	1.526681	1.546639	1.530125	1.553336	1.608050
min	12.753830	11.822230	11.245440	10.711590	10.255130
25%	18.762520	17.506115	16.899070	16.527330	16.281987
50%	19.350015	18.072760	17.459205	17.091615	16.861280
75%	20.079930	18.656610	17.927477	17.593157	17.454690
max	30.960000	30.420980	31.173560	30.562360	28.553240

MODEL DEVELOPMENT PHASE

Feature Selection Report

In the forthcoming update, each feature will be accompanied by a brief description. Users will indicate whether it's selected or not, providing reasoning for their decision. This process will streamline decision-making and enhance transparency in feature selection.

Feature	Description	Selected	Reasoning
objid	Object ID, unique identifier for the observation	No	Used for identification only, not relevant for modeling
specobjid	Spectroscopic object ID	No	Identifier only; does not influence model features
ra	Right Ascension (sky coordinate)	No	Positional data; may not influence intrinsic properties
dec	Declination (sky coordinate)	No	Positional data; not relevant for classification or regression
u	Magnitude in u-band	Yes	Photometric measurement; useful for color/brightness analysis
g	Magnitude in g-band	Yes	Helps define object color, useful for classification
r	Magnitude in r-band	Yes	Important for photometric features
i	Magnitude in i-band	Yes	Adds spectral information
z	Magnitude in z-band	Yes	Complements other band magnitudes
modelFlux_u	Model flux in u-band	No	Flux and magnitude are related; may cause redundancy
modelFlux_g	Model flux in g-band	No	Excluded to avoid multicollinearity with magnitude
modelFlux_r	Model flux in r-band	No	Similar reason as above
modelFlux_i	Model flux in i-band	No	Not chosen to reduce redundancy
modelFlux_z	Model flux in z-band	No	Magnitude already included
petroRad_u	Petrosian radius in u-band	No	May be noisy and inconsistent across bands
petroRad_g	Petrosian radius in g-band	No	Size-related, but not most discriminative
petroRad_i	Petrosian radius in i-band	No	Excluded for simplicity

petroRad_r	Petrosian radius in r-band	No	Not selected due to similar alternatives
petroRad_z	Petrosian radius in z-band	No	Redundant with others
petroFlux_u	Petrosian flux in u-band	No	Flux already represented via magnitude
petroFlux_g	Petrosian flux in g-band	No	Not used to prevent duplicate information
petroFlux_i	Petrosian flux in i-band	No	Avoid redundancy
petroFlux_r	Petrosian flux in r-band	No	Same reason as above
petroFlux_z	Petrosian flux in z-band	No	Flux data excluded
petroR50_u	Petrosian radius at 50% light in u-band	No	Less relevant in classification
petroR50_g	Petrosian radius at 50% light in g-band	No	Similar reasons as above
petroR50_i	Petrosian radius at 50% light in i-band	No	Not chosen for simplicity
petroR50_r	Petrosian radius at 50% light in r-band	No	Excluded to reduce dimensionality
petroR50_z	Petrosian radius at 50% light in z-band	No	Not distinctively informative
psfMag_u	PSF magnitude in u-band	No	One set of magnitudes already included
psfMag_r	PSF magnitude in r-band	No	Avoid mixing photometric systems
psfMag_g	PSF magnitude in g-band	No	Redundant with other magnitude measures
psfMag_i	PSF magnitude in i-band	No	Redundant
psfMag_z	PSF magnitude in z-band	No	Not selected to reduce feature overlap
expAB_u	Axis ratio in exponential model for u-band	Yes	Target variables are depended
expAB_g	Axis ratio in g-band	Yes	Target variables are depended
expAB_r	Axis ratio in r-band	Yes	Target variables are depended
expAB_i	Axis ratio in i-band	Yes	Target variables are depended

expAB_z	Axis ratio in z-band	Yes	Target variables are depended
class	Object class (e.g., STAR, GALAXY, QSO)	No	Because class has only single value
subclass	Subcategory of the main class	Yes	Provides more granular classification information
redshift	Estimated redshift of the object	Yes	Crucial for many astrophysical analyses
redshift_err	Uncertainty in redshift	No	May introduce noise, excluded for simplicity
morphology	Visual morphology category	Yes	High-level feature for physical appearance
agni	Possibly an indicator or flag (domain-specific, unclear without metadata)	Yes	Target variables are dependent

Model Selection Report

Scenario 2:

Model	Performance metric (R Square)
Linear Regression	61.7 %
Decision Tree Regressor	63.2 %
Random Forest Regressor	64.9 %
AdaBoost Regressor	61.4 %
KNN	64.5 %
Stacking	70.8 %

Scenario 1: (The values for in the morphology column are chosen random)

Model	Performance metric (F1 score)
Logistic Regression	30 %
Decision Tree Regressor	29 %
Random Forest Regressor	35 %
AdaBoost Regressor	33 %
KNN	36 %
Stacking	32 %

Scenario 3: (The values for in the agn related column are chosen random)

Model	Performance metric (F1 score)
Logistic Regression	50 %
Decision Tree Regressor	51 %
Random Forest Regressor	52 %
AdaBoost Regressor	52 %
KNN	51 %
Stacking	54 %

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model training Code:

Scenario 2:

LINEAR REGRESSION

```
[ ] linear=LinearRegression()
    lr_model=linear.fit(x_train,y_train)

[ ] y_pred=lr_model.predict(x_test)

[ ] r2_train=lr_model.score(x_train,y_train)
    intercept=lr_model.intercept_
    slope=lr_model.coef_

[ ] r2_test=lr_model.score(x_test,y_test)
    r2_test

    lr_rmse=mean_squared_error(y_test,y_pred)
    lr_mse=mean_squared_error(y_test,y_pred)
    lr_mae=mean_absolute_error(y_test,y_pred)
    lr_mape=mean_absolute_percentage_error(y_test,y_pred)
```

DECISION TREE REGRESSOR

```
[ ] #Buildinag and training Decision tree
    from sklearn import tree

[ ] kf=KFold(n_splits=5,shuffle=True,random_state=42)

[ ] dt=DecisionTreeRegressor()

[ ] param_grid2={"min_samples_split":np.arange(10,12),
                "min_samples_leaf":np.arange(10,12),
                "max_depth":np.arange(6,8)}

[ ] grid_cv2=GridSearchCV(dt,param_grid2,cv=kf,scoring="r2",n_jobs=-1)

[ ] grid_cv2.fit(x_train,y_train)

[ ] grid_cv2.best_score_

[ ] grid_cv2.best_params_

[ ] grid_cv2.best_estimator_

[ ] dt_model=DecisionTreeRegressor(max_depth=4,min_samples_leaf=10,min_samples_split=30)

[ ]
    dt_model.fit(x_train,y_train)

[ ] y_pred1=dt_model.predict(x_test)
```

RANDOM FOREST REGRESSOR

```
#Building and running Random forest regressor
param_grid = {
    "n_estimators": [50,100],
    "min_samples_split": [10],
    "min_samples_leaf": [10],
    "max_depth": [5,7],
}
```

```
rf=RandomForestRegressor()
```

```
grid_cv1=GridSearchCV(rf,param_grid,cv=kf,scoring="r2",n_jobs=-1)
```

```
grid_cv1.fit(x_train,y_train)
```

```
grid_cv1.best_score_
```

```
grid_cv1.best_params_
```

```
grid_cv1.best_estimator_
```

```
rf_model=RandomForestRegressor(max_depth=4,min_samples_leaf=40,min_samples_split=10,n_estimators=10)
```

```
rf_model.fit(x_train,y_train)
```

```
y_pred2=rf_model.predict(x_test)
```

ADA BOOSTER

```
ada=AdaBoostRegressor()
```

```
kf=KFold(n_splits=5)
```

```
#param_grid={"n_estimators":np_arange(10,101,10),
#           "learning_rate":np.arange(0.05,1,0.05),
#}

param_grid={"n_estimators":np.arange(50,101,50),
           "learning_rate":[0.05,0.1,0.2,0.5]
}
```

```
grid_cv=GridSearchCV(ada,param_grid,cv=kf,scoring="r2",n_jobs=-1)
```

```
grid_cv.fit(x_train,y_train)
```

```
ada_model=AdaBoostRegressor(learning_rate=0.05,n_estimators=20,random_state=42)
```

```
ada_model.fit(x_train,y_train)
```

```
y_pred3=ada_model.predict(x_test)
```

```
adr2_train=ada_model.score(x_train,y_train)
r2_score_train.append(adr2_train)
adr2_train
```

KNN

```
#Building and running KNN
r2_scores=[]
for k in range(2,25):
  knn_score=cross_val_score(KNeighborsRegressor(k),x_train,y_train,scoring="r2",cv=kf,n_jobs=-
  r2_scores.append(np.mean(knn_score))
```

```
for k in range(2,25):
  print("number of neighbours:",k,":",r2_scores[k-2])
```

```
plt.figure(figsize=(9,5))
plt.plot(range(2,25),r2_scores,marker="o")
plt.ylabel("r2_scores")
plt.xlabel("k_values")
plt.title("r2_scores in different k values")
plt.xticks(range(0,25,3))
plt.grid()
plt.show()
```

```
k=7
kn_model=KNeighborsRegressor(k).fit(x_train,y_train)
y_pred4=kn_model.predict(x_test)
```

STACKING

```
[ ] #Building an running Stacking
```

```
[ ] level1=[]
    level1.append(("lr",lr_model))
    level1.append(("knn",kn_model))
    level1.append(("svr",SVR()))
    level1.append(("dt",dt_model))
    level1.append(("rnd",rf_model))
    level1.append(("ada",ada_model))
    level2=LinearRegression()
    stack_model=StackingRegressor(estimators=level1,final_estimator=level2,cv=kf,n_jobs=-1)
```

```
[ ] level1
```

```
[ ] st_model=stack_model.fit(x_train,y_train)
    y_pred_st=st_model.predict(x_test)
```

```
[ ] score=cross_val_score(stack_model,x_train,y_train,scoring="r2",cv=kf)
```

```
[ ] print("Rscore:",np.mean(score))
```

```
[ ] str2_train=st_model.score(x_train,y_train)
    r2_score_train.append(str2_train)
    str2_test=st_model.score(x_test,y_test)
    r2_score_test.append(str2_test)
```

Scenario 1:

LOGISTIC REGRESSION(OF MORPHOLOGY)

```
[ ] x = red_df.drop('morphology', axis=1)
    y = red_df['morphology']
```

```
[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

```
[ ] log_model = LogisticRegression(max_iter=1500)  # max_iter helps avoid convergence issues
```

```
[ ] log_model.fit(x_train, y_train)
```

```
[ ] my_pred1 = log_model.predict(x_test)
```

DECISION TREE CLASSIFICATION

```
[ ] mdt = DecisionTreeClassifier(random_state=42)
```

```
[ ] mkf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
[ ] param_grid = {
        'max_depth': [5, 10, 15],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'criterion': ['gini', 'entropy']  # or 'log_loss' for probabilistic output
    }
```

```
[ ] grid_search = GridSearchCV(estimator=mdt,
                               param_grid=param_grid,
                               cv=mkf,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=1)
```

```
[ ] grid_search.fit(x_train, y_train.astype(int))  # Make sure y is integer type
```

```
[ ] grid_search.best_score_
    grid_search.best_params_
    grid_search.best_estimator_
```

```
[ ] best_model = grid_search.best_estimator_
```

```
[ ] my_pred2 = best_model.predict(x_test)
```

RANDOM FOREST CLASSIFICATION

```
[ ] mrf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

```
[ ] param_grid = {
        'n_estimators': [50,100],              # number of trees
        'max_depth': [10],                   # maximum tree depth
        'min_samples_split': [2],             # min samples to split a node
        'min_samples_leaf': [1, 2],             # min samples at a leaf node
        'criterion': ['gini', 'entropy']        # splitting criterion
    }
```

```
[ ] grid_search = GridSearchCV(estimator=mrf,
                               param_grid=param_grid,
                               cv=kf,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=2)
```

```
[165]
    grid_search.fit(x_train, y_train.astype(int))  # Ensure labels are integers
```

```
[166] grid_search.best_score_
      grid_search.best_params_
      grid_search.best_estimator_
```

```
[167] best_model = grid_search.best_estimator_
```

```
[168] my_pred3 = best_model.predict(x_test)
```

ADABOOST CLASSIFICATION

+ Code + Text

```
[171] param_grid = {
          'n_estimators': [50, 100],
          'learning_rate': [0.01, 0.1, 0.5]
      }
```

```
[172] mada=AdaBoostClassifier()
```

```
[174] grid_search = GridSearchCV(
          mada,
          param_grid,
          cv=5,
          scoring='accuracy',
          n_jobs=-1  # Use all CPU cores
      )
      grid_search.fit(x_train, y_train)
```

```
grid_search.best_score_
grid_search.best_params_
grid_search.best_estimator_
```

```
best_model = grid_search.best_estimator_
```

```
my_pred4 = best_model.predict(x_test)
```

KNN

```
mkf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

```
accuracy_scores = []

for k in range(2, 25):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train, cv=mkf, scoring='accuracy')
    accuracy_scores.append(np.mean(scores))
```

+ Code + Text

```
[184] plt.figure(figsize=(10, 5))
      plt.plot(range(2, 25), accuracy_scores, marker='o')
      plt.title("KNN Classification Accuracy for Different k")
      plt.xlabel("k (Number of Neighbors)")
      plt.ylabel("Cross-Validated Accuracy")
      plt.grid(True)
      plt.xticks(range(2, 25, 2))
      plt.show()
```

```
best_k = 14
print(f"Best k: {best_k}")
```

```
m_knn = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
```

```
m_knn.fit(x_train, y_train)
```

```
my_pred5 = m_knn.predict(x_test)
```

STACKING

```
level1 = []
level1.append(("log_model", LogisticRegression(max_iter=1500)))
level1.append(("msvm", SVC()))
level1.append(("mdt", DecisionTreeClassifier()))
level1.append(("mada", AdaBoostClassifier()))
level1.append(("mknn", KNeighborsClassifier()))
```

```
level2 = LogisticRegression()
```

```
mstack_model = StackingClassifier(
    estimators=level1,
    final_estimator=level2,
    cv=3,  # or use StratifiedKFold
    n_jobs=-1
)
```

```
[197] cv_scores = cross_val_score(mstack_model, x_train, y_train, cv=mkf, scoring='accuracy',n_jobs=
      print("Cross-Validated Accuracy Scores:", cv_scores)
      print("Mean Accuracy:", cv_scores.mean())
```

```
mstack_model.fit(x_train, y_train)
```

```
my_pred6 = mstack_model.predict(x_test)
```

Scenario 3:

LOGISTIC REGRESSION(OF AGN IDENTIFICATION)

```
[ ] x = red_df.drop('agni', axis=1)
    y = red_df['agni']

[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

[ ] agnlog_model = LogisticRegression(max_iter=1500)  # max_iter helps avoid convergence issues

[ ] agnlog_model.fit(x_train, y_train)

[ ] agny_pred1 = agnlog_model.predict(x_test)
```

DECISION TREE CLASSIFICATION

```
[ ] agndt = DecisionTreeClassifier(random_state=42)

[ ] agnkf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

[ ] param_grid = {
        'max_depth': [5, 10, 15],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'criterion': ['gini', 'entropy']  # or 'log_loss' for probabilistic output
    }

[ ] grid_search = GridSearchCV(estimator=agndt,
                               param_grid=param_grid,
                               cv=agnkf,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=1)

[ ] grid_search.fit(x_train, y_train.astype(int))  # Make sure y is integer type

[ ] grid_search.best_score_
    grid_search.best_params_
    grid_search.best_estimator_

[ ] best_model = grid_search.best_estimator_

[ ] agny_pred2 = best_model.predict(x_test)
```

RANDOM FOREST CLASSIFICATION

```
[ ] agnrf = RandomForestClassifier(random_state=42, n_jobs=-1)

[ ] param_grid = {
        'n_estimators': [50,100],              # number of trees
        'max_depth': [10],                    # maximum tree depth
        'min_samples_split': [2],              # min samples to split a node
        'min_samples_leaf': [1, 2],              # min samples at a leaf node
        'criterion': ['gini', 'entropy']         # splitting criterion
    }

[ ] grid_search = GridSearchCV(estimator=agnrf,
                               param_grid=param_grid,
                               cv=agnkf,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=2)

[ ]
    grid_search.fit(x_train, y_train.astype(int))  # Ensure labels are integers

[ ] grid_search.best_score_
    grid_search.best_params_
    grid_search.best_estimator_

[ ] best_model = grid_search.best_estimator_

[ ] agny_pred3 = best_model.predict(x_test)
```

ADABOOST CLASSIFICATION

```
[ ] param_grid = {
        'n_estimators': [50, 100],
        'learning_rate': [0.01, 0.1, 0.5]
    }
```

```
[ ] agnada=AdaBoostClassifier()
```

```
[ ] grid_search = GridSearchCV(
        agnada,
        param_grid,
        cv=5,
        scoring='accuracy',
        n_jobs=-1  # Use all CPU cores
    )

    grid_search.fit(x_train, y_train)
```

```
[ ] grid_search.best_score_
    grid_search.best_params_
    grid_search.best_estimator_
```

```
[ ] best_model = grid_search.best_estimator_
```

```
[ ] agny_pred4 = best_model.predict(x_test)
```

KNN

```
[ ] agnkf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

```
[ ] accuracy_scores = []

    for k in range(2, 25):
        agnknn = KNeighborsClassifier(n_neighbors=k,n_jobs=-1)
        scores = cross_val_score(agnknn, x_train, y_train, cv=agnkf, scoring='accuracy')
        accuracy_scores.append(np.mean(scores))
```

```
[ ] plt.figure(figsize=(10, 5))
    plt.plot(range(2, 25), accuracy_scores, marker='o')
    plt.title("KNN Classification Accuracy for Different k")
    plt.xlabel("k (Number of Neighbors)")
    plt.ylabel("Cross-Validated Accuracy")
    plt.grid(True)
    plt.xticks(range(2, 25, 2))
    plt.show()
```

```
[ ] best_k = 10
    print(f"Best k: {best_k}")
```

```
[ ] agn_knn = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
```

```
[ ] agn_knn.fit(x_train, y_train)
```

```
[ ] agny_pred5 = agn_knn.predict(x_test)
```

STACKING

```
[ ] agnlevel1 = []
    agnlevel1.append(("log_model", LogisticRegression(max_iter=1500)))
    agnlevel1.append(("mknn", KNeighborsClassifier()))
    agnlevel1.append(("msvm", SVC()))
    agnlevel1.append(("mdt", DecisionTreeClassifier()))
    agnlevel1.append(("mada", AdaBoostClassifier()))
```

```
[ ] agnlevel2 = LogisticRegression()
```

```
[ ] agnstack_model = StackingClassifier(
        estimators=agnlevel1,
        final_estimator=agnlevel2,
        cv=3,  # or use StratifiedKFold
        n_jobs=-1
    )
```

```
[ ] agncv_scores = cross_val_score(agnstack_model, x_train, y_train, cv=agnkf, scoring='accuracy',
    print("Cross-Validated Accuracy Scores:", agncv_scores)
    print("Mean Accuracy:", agncv_scores.mean())
```

```
[ ] agnstack_model.fit(x_train, y_train)
```

```
[ ] agny_pred6 = agnstack_model.predict(x_test)
```

Model Validation and Evaluation Report:

Scenario 2:

```
[129] rfinal_results=pd.DataFrame()
      for i in range(0,len(model_list)):
        ab=[[model_list[i],r2_score_train[i],r2_score_test[i],rmse[i],mse[i],mae[i],mape[i]]]
        new=pd.DataFrame(ab)
        rfinal_results=pd.concat([rfinal_results,new],axis=0)
      rfinal_results.columns=metric_list
      rfinal_results=rfinal_results.reset_index(drop=True)
      rfinal_results
```

	Models	r2 Score(Train)	r2 Score	RMSE	MSE	MAE	MAPE
0	Linear Regression	0.649842	0.649369	0.011732	0.011732	0.068519	0.747834
1	Decision Tree Regression	0.641418	0.632783	0.011239	0.632783	0.070397	0.711711
2	Random Forest Regressor	0.660115	0.649723	0.010721	0.649723	0.069070	0.709246
3	Adaoost Regressor	0.614396	0.605990	0.012059	0.605990	0.074206	0.802742
4	KNN Regression	0.645589	0.525693	0.014517	0.525693	0.076865	0.813940
5	Stacked Regression	0.723904	0.708576	0.008920	0.708576	0.059873	0.600899

Scenario 1:(For morphology column the data is choosen at random. Hence cannot expect good model validation and evaluation report)

```
                     Models  Accuracy (Train)  Accuracy (Test)  Precision  \
0         Linear Regression          0.342800         0.334500   0.336869
1  Decision Tree Regression          0.341233         0.337367   0.338243
2   Random Forest Regressor          0.343667         0.335933   0.334908
3         Adaoost Regressor          0.341167         0.335800   0.186544
4            KNN Regression          0.333333         0.335300   0.334137
5        Stacked Regression          0.341433         0.333967   0.327692

     Recall  F1-Score
0  0.334500  0.268948
1  0.337367  0.254824
2  0.335933  0.288383
3  0.335800  0.168993
4  0.335300  0.331473
5  0.333967  0.282424
```

Scenario 3:(For AGN related column the data is choosen at random. Hence cannot expect good model validation and evaluation report)

```python
# Create dataframe
agnfinal_results = pd.DataFrame()
for i in range(len(agnmodel_list)):
    row = [[agnmodel_list[i], agnacc_train[i], agnacc_test[i], agnprecision[i], agnrecall[i], agnf1[i]]]
    df = pd.DataFrame(row)
    agnfinal_results = pd.concat([agnfinal_results, df], axis=0)

# Set column names
agnfinal_results.columns = agnmetric_list
agnfinal_results = agnfinal_results.reset_index(drop=True)
print(agnfinal_results)
```

```
                Models  Accuracy (Train)  Accuracy (Test)  Precision  \
0  Logistic Regression          0.500933         0.499167   0.499319
1        Decision Tree          0.498133         0.505900   0.506496
2        Random Forest          0.340267         0.338867   0.337843
3             AdaBoost          0.504300         0.498333   0.248336
4             Stacking          0.497867         0.504200   0.504198
5                  KNN          0.501067         0.503267   0.503430

     Recall  F1-Score
0  0.499167  0.498120
1  0.505900  0.500996
2  0.338867  0.297633
3  0.498333  0.331483
4  0.504200  0.504162
5  0.503267  0.495749
```

MODEL OPTIMIZATION AND TUNING PHASE

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peakperformance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation :

Scenario 2:

Model	Tuned Hypermeters	Optimal values
Decision Tree	`param_grid2={"min_samples_split":np.arange(10,12),` `"min_samples_leaf":np.arange(10,12),` `"max_depth":np.arange(6,8)}`	`grid_cv2.best_params_` `{'max_depth': np.int64(7` `'min_samples_leaf': np.` `'min_samples_split': np`
Random Forest	`param_grid = {` `"n_estimators": [50,100],` `"min_samples_split": [10],` `"min_samples_leaf": [10],` `"max_depth": [5,7],` `}`	`grid_cv1.best_params_` `{'max_depth': 7,` `'min_samples_leaf':` `'min_samples_split':` `'n_estimators': 100}`

Scenario 1:

Model	**Tuned Hypermeters**	**Optimal values**
Decision Tree	`param_grid = {` `'max_depth': [5, 10, 15],` `'min_samples_split': [2, 5, 10],` `'min_samples_leaf': [1, 2, 4],` `'criterion': ['gini', 'entropy']  # or 'log_loss' for probabilistic output` `}`	`grid_search.best_params_` `{'learning_rate': 0.01, 'n_estimators': 50}`
Random forest	`[162] param_grid = {` `'n_estimators': [50,100],  # number of trees` `'max_depth': [10],  # maximum tree depth` `'min_samples_split': [2],  # min samples to split a node` `'min_samples_leaf': [1, 2],  # min samples at a leaf node` `'criterion': ['gini', 'entropy']  # splitting criterion` `}`	`[288] grid_search.best_params_` `{'learning_rate': 0.01, 'n_estimators': 50}`
Adaboost		

	``[171] param_grid = { 'n_estimators': [50, 100], 'learning_rate': [0.01, 0.1, 0.5] }```	```[289] grid_search.best_params_ {'learning_rate': 0.01, 'n_estimators': 50}```

Scenario 3:

Model	Tuned Hypermeters	Optimal values
Decision Tree	```[224] param_grid = { 'max_depth': [5, 10, 15], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'criterion': ['gini', 'entropy'] # or 'log_loss' for probabilistic output }```	```[290] grid_search.best_params_ {'learning_rate': 0.01, 'n_estimators': 50}```
Random forest	```param_grid = { 'n_estimators': [50,100], 'max_depth': [10], 'min_samples_split': [2], 'min_samples_leaf': [1, 2], 'criterion': ['gini', 'entropy'] }```	```[288] grid_search.best_params_ {'learning_rate': 0.01, 'n_estimators': 50}```

Performance Metric:

Scenario 2:

```
[129] rfinal_results=pd.DataFrame()
      for i in range(0,len(model_list)):
        ab=[[model_list[i],r2_score_train[i],r2_score_test[i],rmse[i],mse[i],mae[i],mape[i]]]
        new=pd.DataFrame(ab)
        rfinal_results=pd.concat([rfinal_results,new],axis=0)
      rfinal_results.columns=metric_list
      rfinal_results=rfinal_results.reset_index(drop=True)
      rfinal_results
```

	Models	r2 Score(Train)	r2 Score	RMSE	MSE	MAE	MAPE
0	Linear Regression	0.649842	0.649369	0.011732	0.011732	0.068519	0.747834
1	Decision Tree Regression	0.641418	0.632783	0.011239	0.632783	0.070397	0.711711
2	Random Forest Regressor	0.660115	0.649723	0.010721	0.649723	0.069070	0.709246
3	Adaoost Regressor	0.614396	0.605990	0.012059	0.605990	0.074206	0.802742
4	KNN Regression	0.645589	0.525693	0.014517	0.525693	0.076865	0.813940
5	Stacked Regression	0.723904	0.708576	0.008920	0.708576	0.059873	0.600899

Scenario 3:

```
# Create dataframe
agnfinal_results = pd.DataFrame()
for i in range(len(agnmodel_list)):
    row = [[agnmodel_list[i], agnacc_train[i], agnacc_test[i], agnprecision[i], agnrecall[i], agnf1[i]]]
    df = pd.DataFrame(row)
    agnfinal_results = pd.concat([agnfinal_results, df], axis=0)

# Set column names
agnfinal_results.columns = agnmetric_list
agnfinal_results = agnfinal_results.reset_index(drop=True)
print(agnfinal_results)
```

```
                Models  Accuracy (Train)  Accuracy (Test)  Precision  \
0  Logistic Regression          0.500933         0.499167   0.499319
1        Decision Tree          0.498133         0.505900   0.506496
2        Random Forest          0.340267         0.338867   0.337843
3             AdaBoost          0.504300         0.498333   0.248336
4             Stacking          0.497867         0.504200   0.504198
5                  KNN          0.501067         0.503267   0.503430

     Recall  F1-Score
0  0.499167  0.498120
1  0.505900  0.500996
2  0.338867  0.297633
3  0.498333  0.331483
4  0.504200  0.504162
5  0.503267  0.495749
```

Scenario1:

```
# Create dataframe
mfinal_results = pd.DataFrame()
for i in range(len(model_list)):
    row = [[model_list[i], macc_train[i], macc_test[i], mprecision[i], mrecall[i], mf1[i]]]
    df = pd.DataFrame(row)
    mfinal_results = pd.concat([mfinal_results, df], axis=0)

# Set column names
mfinal_results.columns = mmetric_list
mfinal_results = mfinal_results.reset_index(drop=True)
print(mfinal_results)
```

```
                     Models  Accuracy (Train)  Accuracy (Test)  Precision  \
0         Linear Regression          0.342800         0.334500   0.336869
1  Decision Tree Regression          0.341233         0.337367   0.338243
2   Random Forest Regressor          0.343667         0.335933   0.334908
3         Adaboost Regressor          0.341167         0.335800   0.186544
4            KNN Regression          0.333333         0.335300   0.334137
5        Stacked Regression          0.341433         0.333967   0.327692

     Recall  F1-Score
0  0.334500  0.268948
1  0.337367  0.254824
2  0.335933  0.288383
3  0.335800  0.168993
4  0.335300  0.331473
5  0.333967  0.282424
```