Jacob Goldstein COP 4773

HW 4

```
#Step 1. PROBLEM UNDERSTANDING
# Based on stellar classification dataset, predict object to be either a star,
galaxy or quasar

#1.1. Dataset selection:
# Name: Stellar Classification Dataset - SDSS17
# Description: The data consists of 100,000 observations of space taken by the
SDSS (Sloan Digital Sky Survey).
#Every observation is described by 17 feature columns and 1 class column which
identifies it to be either a star, galaxy or quasar.
# Link:
https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17?resource=dow

# Number of Instances: 100000

# Number of Attributes: 17

# Attribute Information:

#obj_ID = Object Identifier, the unique value that identifies the object in the
image catalog used by the CAS
#alpha = Right Ascension angle (at J2000 epoch)
#delta = Declination angle (at J2000 epoch)
#u = Ultraviolet filter in the photometric system
#g = Green filter in the photometric system
#r = Red filter in the photometric system
#i = Near Infrared filter in the photometric system
#z = Infrared filter in the photometric system
#rereun_ID = Rerun Number to specify how the image was processed
#cam_col = Camera column to identify the scanline within the run
#field_ID = Field number to identify each field
#spec_obj_ID = Unique ID used for optical spectroscopic objects (this means that
2 different observations with the same spec_obj_ID must share the output class)
#class = object class (galaxy, star or quasar object)
#redshift = redshift value based on the increase in wavelength
#plate = plate ID, identifies each plate in SDSS
#MJD = Modified Julian Date, used to indicate when a given piece of SDSS data was
taken
#fiber_ID = fiber ID that identifies the fiber that pointed the light at the
focal plane in each observation
# class: class_0, class_1, class_2

#1.2. Defining a classification task.
# TASK 1: Based on the features or predictors classify the celestial obect in
three classes (0, 1, 2).
```

```
# TASK 2: Divide the redshift in two categories, high and low, and try to
classify the celestial object based on other predictors.

#1.3. Defining a prediction task. (OPTIONAL)
# TASK 3: Predict one of the features (for example redshift) based on the others.
```

```
#Step 2. DATA PREPARATION

# 2.1. Load the dataset

# 2.1.1 Load required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.discrete.discrete_model import Logit

from sklearn import datasets
from sklearn.linear_model import LinearRegression, Lasso, LassoCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_score, train_test_split, KFold
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import silhouette_score, davies_bouldin_score,
mean_squared_error, r2_score
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.pipeline import Pipeline

#2.1.2 Load the wine dataset
# Load from sklearn
data = pd.read_csv('star_classification.csv')


# Show the first few rows of the DataFrame
data.head()
```

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	re
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	3606	3(
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	4518	3(
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	3606	3(
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	4192	3(
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	8102	3(

```
# Show feature types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   obj_ID       100000 non-null  float64
 1   alpha        100000 non-null  float64
 2   delta        100000 non-null  float64
 3   u            100000 non-null  float64
 4   g            100000 non-null  float64
 5   r            100000 non-null  float64
 6   i            100000 non-null  float64
 7   z            100000 non-null  float64
 8   run_ID       100000 non-null  int64
 9   rerun_ID     100000 non-null  int64
 10  cam_col      100000 non-null  int64
 11  field_ID     100000 non-null  int64
 12  spec_obj_ID  100000 non-null  float64
 13  class        100000 non-null  object
 14  redshift     100000 non-null  float64
 15  plate        100000 non-null  int64
 16  MJD          100000 non-null  int64
 17  fiber_ID     100000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
```

```
# 2.2. Handling missing data.

# Creating features or encoding categorical values.
# Applying feature scaling or normalization.

# Map 'Good' to 1 and 'Bad' to 0 in the 'Quality' column
data['class'] = data['class'].map({'GALAXY': 0, 'STAR': 1, 'QSO': 2})
```

```
# Show the first few rows of the DataFrame with updated class values
data.head()
```

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	re
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	3606	3(
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	4518	3(
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	3606	3(
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	4192	3(
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	8102	3(

```
# Show feature types of new class info
data.info()
```

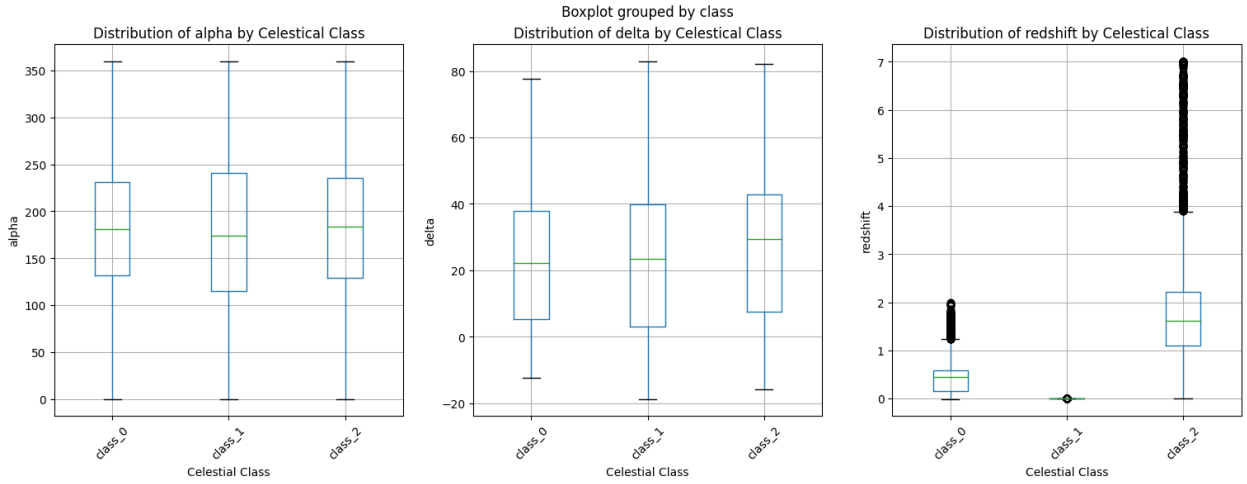
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   obj_ID       100000 non-null  float64
 1   alpha        100000 non-null  float64
 2   delta        100000 non-null  float64
 3   u            100000 non-null  float64
 4   g            100000 non-null  float64
 5   r            100000 non-null  float64
 6   i            100000 non-null  float64
 7   z            100000 non-null  float64
 8   run_ID       100000 non-null  int64  
 9   rerun_ID     100000 non-null  int64  
 10  cam_col      100000 non-null  int64  
 11  field_ID     100000 non-null  int64  
 12  spec_obj_ID  100000 non-null  float64
 13  class        100000 non-null  int64  
 14  redshift     100000 non-null  float64
 15  plate        100000 non-null  int64  
 16  MJD          100000 non-null  int64  
 17  fiber_ID     100000 non-null  int64  
dtypes: float64(10), int64(8)
memory usage: 13.7 MB
```

```
#Step 3. EXPLORATORY DATA ANALYSIS

#3.1 Analyzing basic trend statistics, generating box plots, scatter plots, among
others.
```

```
#3.1.1. Trend statistics
data.describe()
```

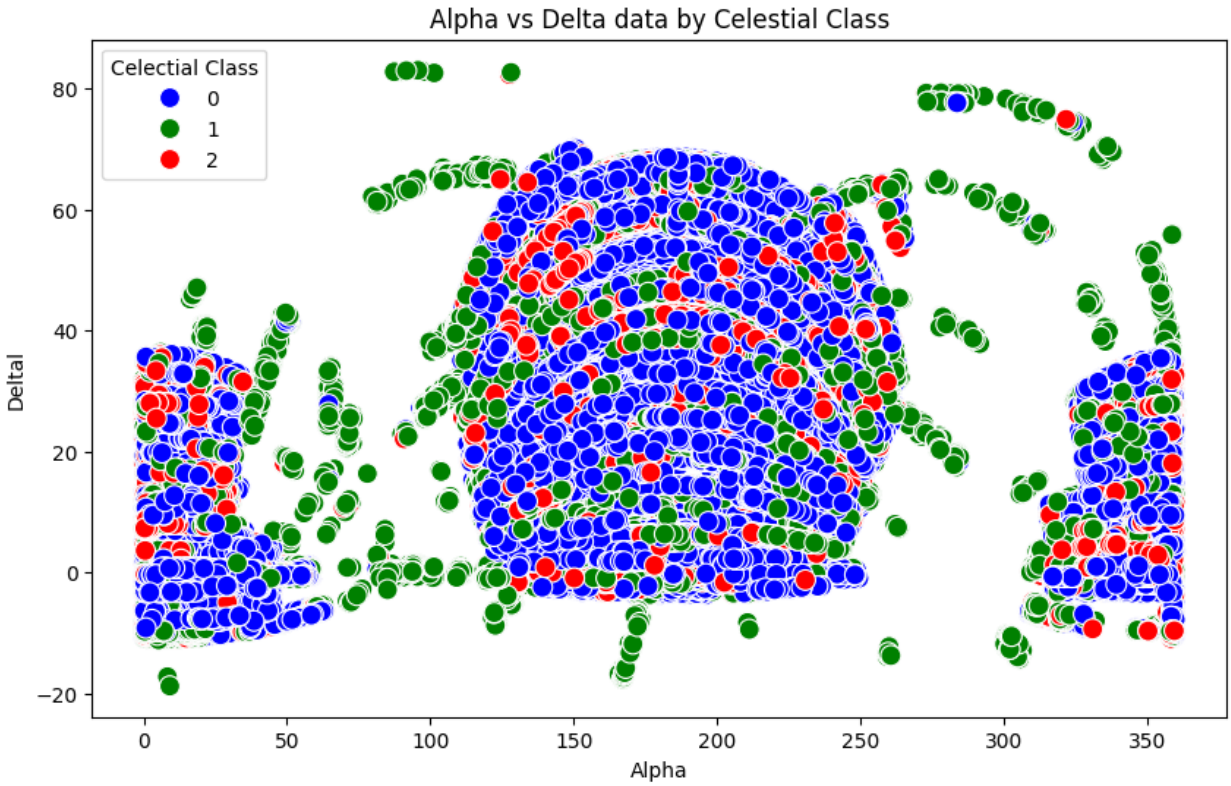
	obj_ID	alpha	delta	u	g	r	i
count	1.000000e+05	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	1000
mean	1.237665e+18	177.629117	24.135305	21.980468	20.531387	19.645762	19.08
std	8.438560e+12	96.502241	19.644665	31.769291	31.750292	1.854760	1.757
min	1.237646e+18	0.005528	-18.785328	-9999.000000	-9999.000000	9.822070	9.469
25%	1.237659e+18	127.518222	5.146771	20.352353	18.965230	18.135828	17.73
50%	1.237663e+18	180.900700	23.645922	22.179135	21.099835	20.125290	19.40
75%	1.237668e+18	233.895005	39.901550	23.687440	22.123767	21.044785	20.39
max	1.237681e+18	359.999810	83.000519	32.781390	31.602240	29.571860	32.14

```
# 3.1.2. Box Plots

X = data.loc[:, data.columns != "class"]     # Select all columns except the one
you will predict
X = sm.add_constant(X)
y = data['class']

# Box plot for three columns in the DataFrame
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# List of predictors to plot
predictors = ['alpha', 'delta', 'redshift']
target_names = ['class_0','class_1','class_2']

# Boxplot iteration over the predictors
for ax, predictor in zip(axes, predictors):
    data.boxplot(column=[predictor], by='class', ax=ax, grid=True)
    ax.set_title(f'Distribution of {predictor} by Celestical Class')
    ax.set_xlabel('Celestial Class')
    ax.set_ylabel(f'{predictor}')
    ax.set_xticklabels(target_names, rotation=45)  # Rotate class names for
better visibility

plt.show()
```

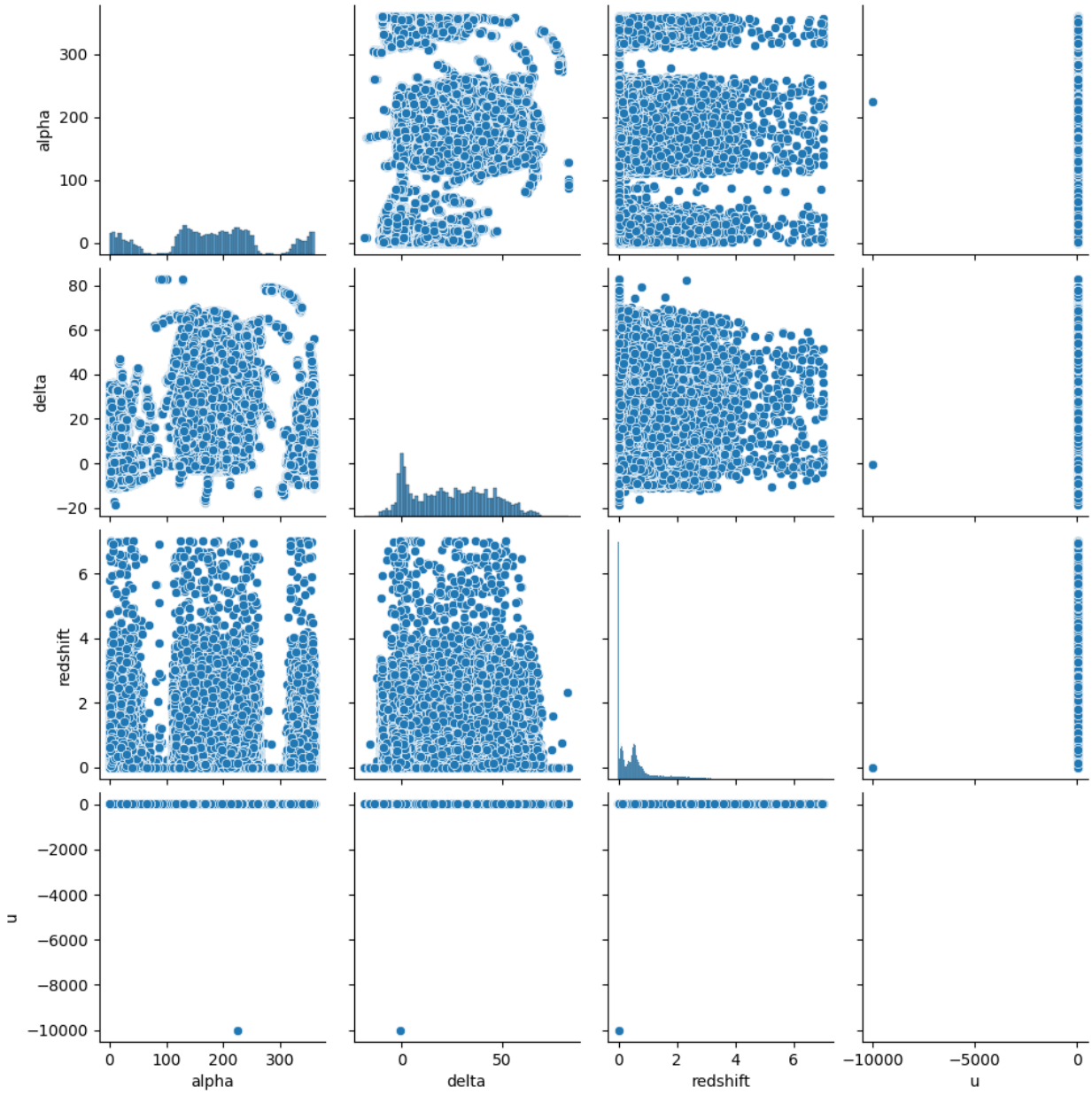


#ANALYSIS: NEED 2 UPDATE The boxplots illustrate the distribution of 'alpha,' 'delta,' and 'redshift' across three different celestial classes. For 'alpha,' all classes are similar. Class_1 has slightly larger variance, inidicating slightly larger variability. In the 'delta' plot, class_2 has the largest median, indicating it has more celestial objects with a higer delta value. Lastly, 'redshift' content contains the most outliers. Class_1 does not even have a boxplot. This signifies there is no variance, or the data is too similar. These distributions offer useful insights into the celestial profile characteristics that could potentially distinguish celestial varieties.

```
#3.1.2. Scatter plot between two predictors

plt.figure(figsize=(10, 6))
palette = {0: 'blue', 1: 'green', 2: 'red'} # Define custom colors for each class
sns.scatterplot(x='alpha', y='delta', hue='class', data=data, palette=palette,
s=100)
plt.title('Alpha vs Delta data by Celestial Class')
plt.xlabel('Alpha')
plt.ylabel('Deltal')
plt.legend(title='Celectial Class')
plt.show()
```




```
#3.1.3. Scatter Plot Matrix with histograms
sns.pairplot(data[['alpha','delta','redshift','u']], diag_kind='hist')
plt.show()
```




```
#3.2. Conducting correlation and multicollinearity analysis

#3.2.1. Variance Inflation Factor
def calculate_vif(dataframe):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = dataframe.columns
    vif_data["VIF"] = [variance_inflation_factor(dataframe.values, i)
                       for i in range(dataframe.shape[1])]
    return vif_data

df_features = data.drop(columns=['obj_ID', 'run_ID', 'rerun_ID', 'cam_col',
'field_ID', 'spec_obj_ID', 'class', 'fiber_ID', 'plate', 'MJD'])
```

```
print(calculate_vif(df_features))
```

```
    Feature          VIF
0     alpha     4.386051
1     delta     2.539445
2         u  1140.594553
3         g  3754.680991
4         r  3713.074950
5         i  3527.111535
6         z  2011.448255
7  redshift     1.933105
```

#ANALYSIS: NEED 2 UPDATE Results indicate significant multicollinearity, particularly with g (VIF = 3754.68), r (VIF = 3713.08), and i (VIF = 3527.11), suggesting they are highly correlated with other predictors. This correlation can lead to unreliable regression coefficients. Z (VIF = 2011.45), and u (VIF = 1140.59) are also significantly large, pointing to notable multicollinearity but to a lesser extent. Other variables like alpha, delta, and redshift indicate moderate multicollinearity.

```
#Step 4. SETUP PHASE
#4.1. Preparing the Python environment, importing libraries. - Done in 2.2.1.
#4.2. Setting up your X (features) and Y (target) data. - Done in 3.1.2.
#4.3. Split the dataset into training and testing sets for validation - (Done
later on each model)
```

```
#Step 5. MODELING PHASE
#Apply several models:
#5.1.   Multiple Linear Regression   (TASK 3)
#5.2.   Multiple Logistic Regression (TASK 2)
#5.3.   Linear Regression using Regularization. (TASK 3)
#5.4.   Polynomial Regression (TASK 3)
#5.5.   LDA - Linear Discriminant Analysis (TASK 1)
#5.6.   Decision Tree for classification (TASK 1)
#5.7.   Decision Tree for regression (TASK 3)
#5.8.   Random Forest Classifier (TASK 1)
#5.9.   Support Vector Machines (TASK 2)
#5.10.  K-means - Unsupervised Learning (TASK 1)
```

```
# FOR THE REGRESSION TASK
#5.1.   Multiple Linear Regression   (TASK 3)

#5.1.1. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['redshift']
```

```
#5.1.2. Split the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.1.3. Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

#5.1.4. Train a linear regression model
linear_reg = LinearRegression()
linear_reg.fit(X_scaled, y_train)

#5.1.5. Standardizing the test set
X_test_scaled = scaler.transform(X_test)

#5.1.6. Predicting 'alcohol' levels using the trained model
y_pred = linear_reg.predict(X_test_scaled)

#5.1.7. Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

#5.1.8. Display the model statistics
print(f"Coefficients: {linear_reg.coef_}")
print(f"Intercept: {linear_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Coefficients: [ 0.         -2.17197338  2.16891917  0.01105015  0.34143148]
Intercept: 0.5772186994596289
Mean Squared Error (MSE): 0.3979112107658387
R-squared (R2): 0.24741442203063813
```

```
#5.1. ANALYSIS
#The Multiple Linear Regression model used to predict redshift levels from the
selected celestial characteristics yielded
#a Mean Squared Error (MSE) of approximately 0.399, which suggests that the
predictions are, on average, relatively
#close to the actual values. However, there is still some variance that the model
does not account for. The R-squared
#value of 0.247 indicates that about 24.7% of the variance in redshift levels is
explained by the model. This level
#of explanation suggests a below moderate fit, where the chosen features have a
unsignificant, but not exclusive, influence on
#the redshift content. The model appears to capture the general trend in the
data, yet there's room for improvement,
```

```
#possibly by feature engineering, inclusion of additional predictors, or
employing more complex modeling techniques.
```

```
# FOR THE CLASSIFICATION TASK
#5.2.   Multiple Logistic Regression (TASK 2)

#5.2.1. Create a binary category
data['redshift_level'] = 0 # Create a new column 'redshift_level' initialized
with zeros
data.loc[data['redshift'] > data['redshift'].mean(), 'redshift_level'] = 1  # Set
'redshift_level' to 1 when 'redshift' is above the mean

#5.2.2. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['redshift_level']

#5.2.3. Split the dataset in training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.2.4. Fit a logistic regresion model
model = sm.Logit(y_train, X_train).fit()

#5.2.5. Display model summary
model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.402388
         Iterations 9
```

Dep. Variable:	redshift_level	**No. Observations:**	80000
Model:	Logit	**Df Residuals:**	79995
Method:	MLE	**Df Model:**	4
Date:	Sun, 21 Apr 2024	**Pseudo R-squ.:**	0.3738
Time:	19:00:52	**Log-Likelihood:**	-32191.
converged:	True	**LL-Null:**	-51410.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	-21.7151	0.195	-111.255	0.000	-22.098	-21.333
u	-0.5135	0.009	-54.720	0.000	-0.532	-0.495
g	0.5411	0.014	38.795	0.000	0.514	0.568
alpha	-0.0003	9.94e-05	-3.002	0.003	-0.000	-0.000
i	1.0840	0.012	90.703	0.000	1.061	1.107

```
#5.2.6. Predicting probabilities
y_pred_prob = model.predict(X_test)

#5.2.7. Convert probabilities to binary predictions based on a threshold of 0.5
y_pred = (y_pred_prob > 0.5).astype(int)

#5.2.8. Predicting and evaluating the model
print(classification_report(y_test.values, y_pred.values))
```

```
              precision    recall  f1-score   support

           0       0.85      0.87      0.86     13205
           1       0.73      0.69      0.71      6795

    accuracy                           0.81     20000
   macro avg       0.79      0.78      0.78     20000
weighted avg       0.81      0.81      0.81     20000
```

```
#ANALYSIS:
#The logistic regression model performed moderatley successfully on the
classification task of predicting alcohol_level with
#an overall accuracy of 81%, as reflected in the precision-recall metrics.
#The model's Pseudo R-squared value of 0.3738 indicates a moderate explanatory
power, and the Log-Likelihood Ratio (LLR)
#p-value of 0 suggests that the model as a whole is statistically significant and
the null model is rejected.
#The coefficients for u, g, alpha, and i were significant predictors, with alpha
showing
#the largest p-value of 0.030.
#In practical terms, the model's ability to differentiate between the classes is
good, with a precision of 0.73 and a
#recall of 0.69 for the higher redshift level class (1), and better precision and
recall
#for the lower redshift level class (0). These results suggest that the selected
features,
#are effective at predicting redshift levels in stars, galazxies, and quasars,
and the model is moderatley-calibrated and reliable
#for making predictions within this dataset.
```

```
#5.3.   Linear Regression using Regularization. (TASK 3)

#5.3.1. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['redshift']
```

```
#5.3.2. Splitting the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.3.3. Standardizing the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

#5.3.4. Standardizing the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).flatten()
# Reshape for a single feature

#5.3.5. Training a Lasso regression model using some alpha
lasso_reg = Lasso(alpha=0.05)
lasso_reg.fit(X_train_scaled, y_train_scaled)

#5.3.6. Predicting 'alcohol' levels using the trained Lasso regression model
y_pred_scaled = lasso_reg.predict(X_test_scaled)

#5.3.7. Reversing the scaling of predictions to original scale
y_pred_lasso = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).flatten()

#5.3.8. Evaluating the model's performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

#5.3.9. Displaying the model statistics
print("Lasso Regression Model")
print(f"Coefficients: {lasso_reg.coef_}")
print(f"Intercept: {lasso_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"R-squared (R2): {r2_lasso}")
```

```
Lasso Regression Model
Coefficients: [ 0.         -0.         -0.          0.          0.44403807]
Intercept: 3.7706146064885623e-16
Mean Squared Error (MSE): 0.4047726817504748
R-squared (R2): 0.23443704424639866
```

```
#ANALYSIS
#The Lasso Regression model, with an alpha value of 0.05, resulted in a model
where some coefficients were reduced to zero,
#or completely eliminated, indicating that the corresponding features were deemed
less important by the regularization process.
```

```
#The model's intercept is approximately 57.72, serving as a baseline prediction
when
#all features are at their mean values. The Mean Squared Error (MSE) is roughly
0.234, which suggests that the model's
#predictions are reasonably close to the actual alcohol levels, and the R-squared
(R2) value is 0.234, indicating
#that nearly 23.4% of the variability in the response variable is explained by
the model. This performance is comparable
#to the non-regularized Multiple Linear Regression model, indicating that Lasso's
feature selection did not significantly
#change the predictive capability in this case.
```

```
#5.4.   Polynomial Regression (TASK 3)

#5.4.1. Selecting a subset of the features and target
X = data[['u', 'g', 'alpha', 'i']]
y = data['redshift']

#5.4.2. Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.4.3. Using a pipeline for 4th-degree polynomial regression
poly4_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=4)),  # Then, generate polynomial features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

#5.4.4. Fit the pipeline on the training data
poly4_pipeline.fit(X_train, y_train)

#5.4.5.Predict on the test data
y_pred_poly4 = poly4_pipeline.predict(X_test)

#5.4.6.Calculate MSE
mse_poly4 = mean_squared_error(y_test, y_pred_poly4)

#5.4.7.Using a pipeline for 3rd-degree polynomial regression
poly3_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=3)),  # Then, generate polynomial features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

#5.4.8.Fit the pipeline on the training data
```

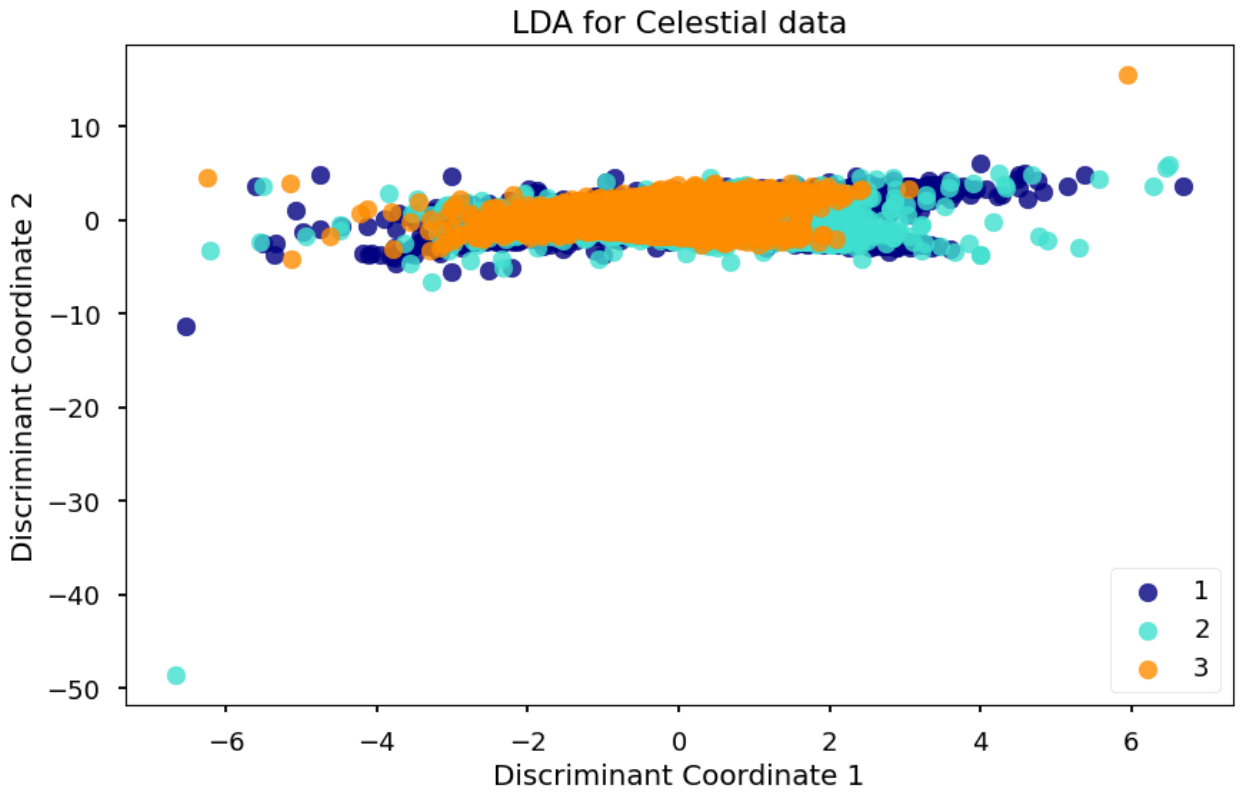
```
poly3_pipeline.fit(X_train, y_train)

#5.4.9.Predict on the test data
y_pred_poly3 = poly3_pipeline.predict(X_test)

#5.4.10.Calculate MSE
mse_poly3 = mean_squared_error(y_test, y_pred_poly3)

#5.4.11.Print the MSE for both models
print(f"Mean Squared Error for 4th-degree Polynomial: {mse_poly4:.2f}")
print(f"Mean Squared Error for 3rd-degree Polynomial: {mse_poly3:.2f}")
```

```
Mean Squared Error for 4th-degree Polynomial: 0.36
Mean Squared Error for 3rd-degree Polynomial: 0.37
```

```
#ANALYSIS
#The polynomial regression models displayed significant similiarites in
performance, with the 4th-degree and 3rd-degree
#models demonstrating good fits and generalization, achieving a low MSE of 0.36,
and 0.37, resepectively.
#When compared to the Lasso regression, which had an MSE of approximately
#0.405 and lower predictive accuracy, it's evident that the Lasso model is
slightly worse in this context.
#The 4th degree polynomial is a more reliable choice for this dataset than the
Lasso regression.
```

```
#5.5. LDA - Linear Discriminant Analysis (TASK 1)

#5.5.2. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['class']

#5.5.3. Split the dataset in training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.5.4. Fit an LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

#5.5.5. Transform the data using the fitted LDA in the new space
X_r_lda = lda.transform(X_train)

#5.5.6. Set the style and create the figure
```

```
target_names = ['1','2','3']     # Your classes name
with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(figsize=[10,6])
    colors = ['navy', 'turquoise', 'darkorange']
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):
        ax.scatter(X_r_lda[y_train == i, 0], X_r_lda[y_train == i, 1], alpha=.8,
label=target_name, color=color)
    ax.set_title('LDA for Celestial data')
    ax.set_xlabel('Discriminant Coordinate 1')
    ax.set_ylabel('Discriminant Coordinate 2')
    ax.legend(loc='best')
    plt.show()
```




```
#5.5.7. Make predictions on the test set
y_pred = lda.predict(X_test)

#5.5.8. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

#5.5.9. Print detailed classification metrics
print(classification_report(y_test, y_pred))
```

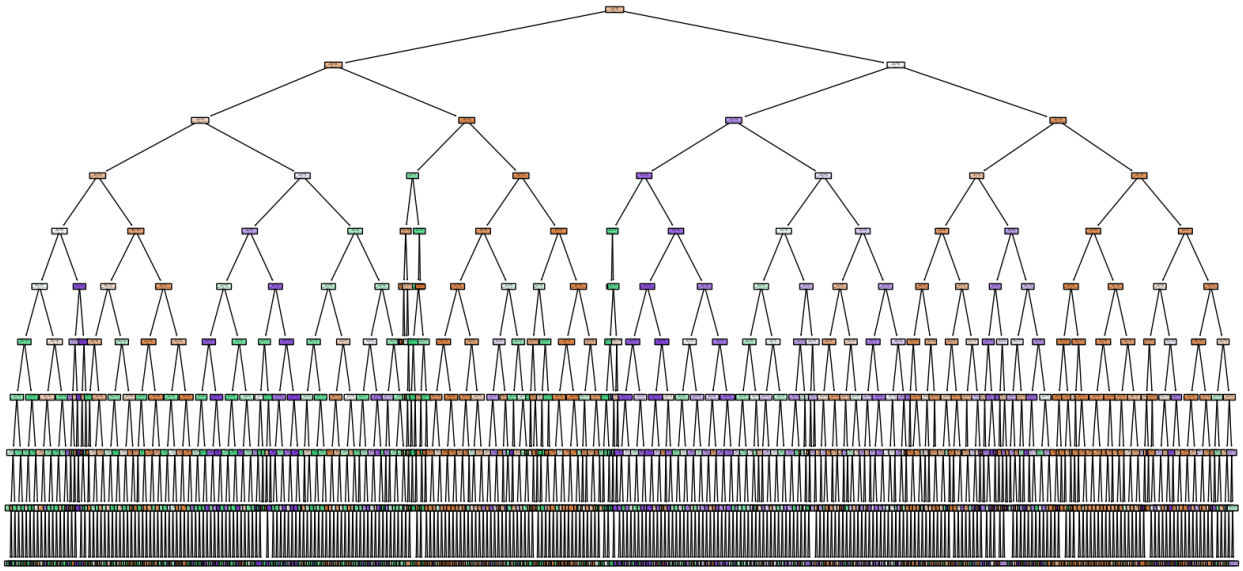
```
Accuracy: 0.59875
              precision    recall  f1-score   support

           0       0.62      0.90      0.73     11860
           1       0.00      0.00      0.00      4343
           2       0.47      0.34      0.40      3797

    accuracy                           0.60     20000
   macro avg       0.36      0.41      0.38     20000
weighted avg       0.46      0.60      0.51     20000
```

```
#ANALYSIS:
#The Linear Discriminant Analysis (LDA) model demonstrated acceptable performance
in classifying the test set with an
#accuracy of 59.9%. The results indicate varying precision, recall, and F1-scores
across all classes. Notably, class 1 recieved 0%
#in all three categories, suggesting that the model could not effectively
differentiate between the three classes.
#Average and bad metrics across all categories conclude that LDA is not the best
at handling this models metrics.
```

```
#5.6.   Decision Tree for classification (TASK 1)

#5.6.1. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
#X = sm.add_constant(X)  # Adds a constant column to input features to account
for the intercept
y = data['class']
#5.6.2. Split the dataset in training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.6.3. Creating a Decision Tree Classifier
dtc = DecisionTreeClassifier(max_depth=10, random_state=42)

#5.6.4. Training the model
dtc.fit(X_train, y_train)

#5.6.5. Making predictions on the test data
y_pred = dtc.predict(X_test)

#5.6.6. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

#5.6.9. Print detailed classification metrics
print(classification_report(y_test, y_pred))
```

```
#5.6.10 Plotting the Decision Tree
feature_names = ['u', 'g', 'alpha', 'i']
class_names = ['1','2','3']
plt.figure(figsize=(20,10))
plot_tree(dtc, filled=True, feature_names=feature_names, class_names =
class_names, rounded=True)
plt.show()
```

```
Accuracy: 0.806
              precision    recall  f1-score   support

           0       0.85      0.91      0.88     11860
           1       0.67      0.57      0.62      4343
           2       0.77      0.76      0.77      3797

    accuracy                           0.81     20000
   macro avg       0.77      0.74      0.75     20000
weighted avg       0.80      0.81      0.80     20000
```

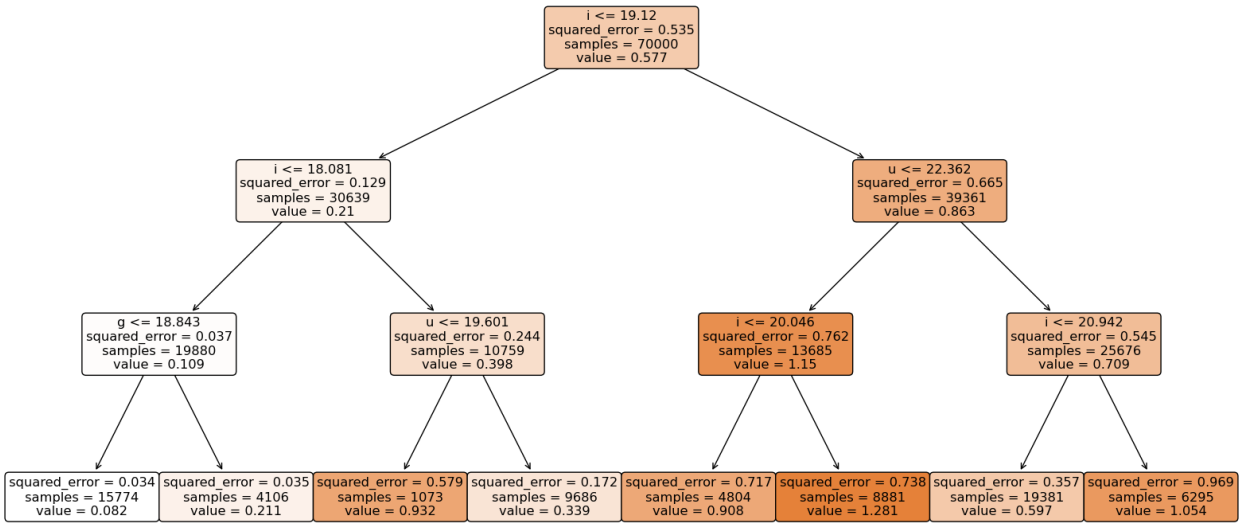


```
#ANALISIS
#The decision tree classifier achieved a good accuracy of 80.6% on the test set,
reflecting a strong overall performance
#in classifying the celestial dataset into three categories. The precision,
recall, and F1-scores varied across
#all classes. Specifically, class 0 had the largest precision and recall at 0.85
and 0.91, indicating strong model sensitivity
#and prediction accuracy for this class. Class 2 was slightly lower, and class 1
was slighlty lower than class 2.
```

```
#The average recall of 0.57 for class 1 suggests room for improvement in
identifying
#all true positives in this category.
#The weighted and macro averages nearing 0.8 for all metrics represent the
model's capability
#to generalize well across stars, galaxies, and quasars. These results suggest
that the decision tree model, with its current
#configuration, is effective, though there is be an opportunity to fine-tune it
to improve precision and recall for class 1 and 2
#without compromising other areas.
```

```
#5.7.   Decision Tree for regression (TASK 3)

#5.3.1. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
#X = sm.add_constant(X)  # Adds a constant column to input features to account
for the intercept
y = data['redshift']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Creating a Decision Tree Regressor
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)

# Training the model
regressor.fit(X_train, y_train)
feature_names = ['u', 'g', 'alpha', 'i']

# Plotting the Decision Tree
plt.figure(figsize=(20,10))
plot_tree(regressor, filled=True, feature_names=feature_names, rounded=True)
plt.show()

# Predicting the alcohol levels in the test set
y_pred = regressor.predict(X_test)

# Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
# R-squared Value
r2 = r2_score(y_test, y_pred)

# Displaying the statistics
print(f"Decision Tree Regressor")
print(f"Mean Squared Error (MSE): {mse}")
```

```
print(f"R-squared (R2): {r2}")
```




```
Decision Tree Regressor
Mean Squared Error (MSE): 0.37302131205343236
R-squared (R2): 0.2977470781957675
```

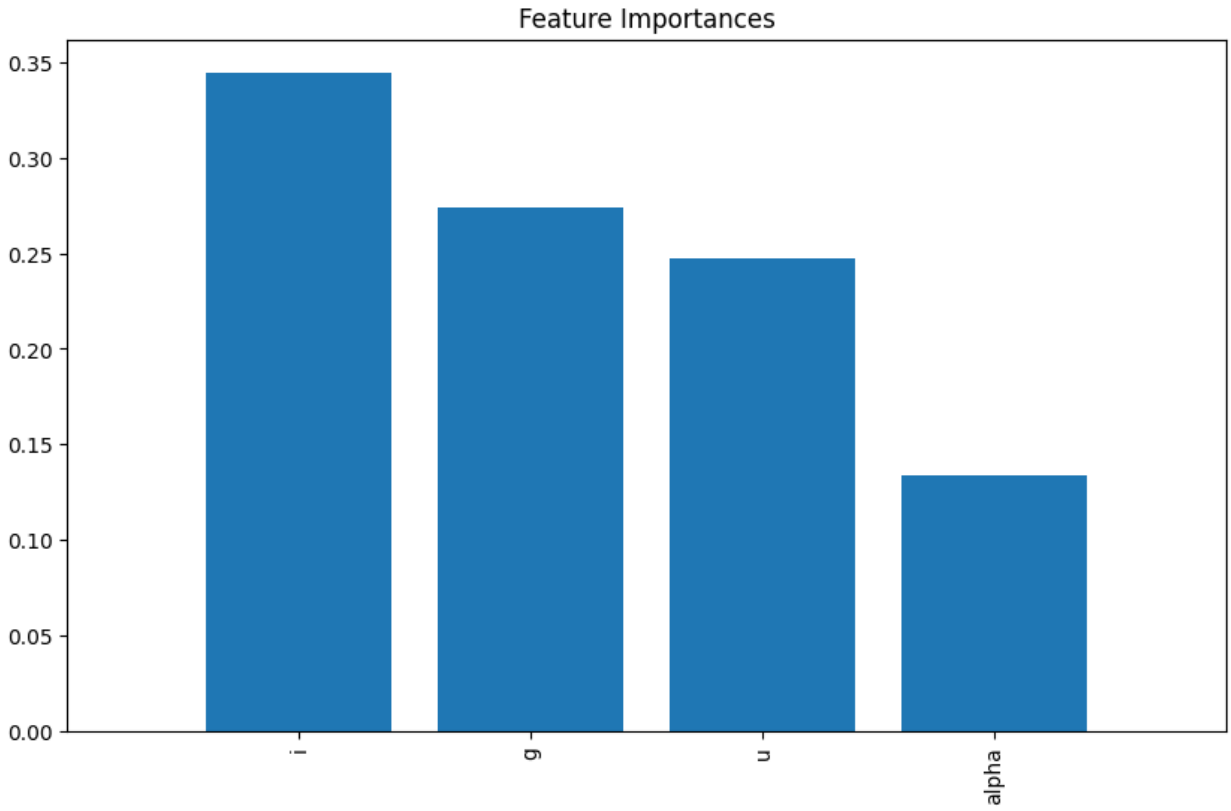
```
#ANALYSIS
#The Decision Tree Regressor, set to a maximum depth of 3, yielded a Mean Squared
Error (MSE) of approximately 0.373,
#indicating the predictions are generally within a reasonable range of the actual
values, but with room for improvement.
#The R-squared (R2) value of about 0.298 suggests that the model explains less
than half of the variance in the target variable,
#which is less than moderate predictive performance. This model, while not as
accurate as previous models like the Lasso Regression,
#still provides valuable insights and could benefit from further tuning or
integration with ensemble methods to
#improve prediction accuracy.
```

```
#5.8.   Random Forest Classifier (TASK 1)

#5.8.1. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
#X = sm.add_constant(X)  # Adds a constant column to input features to account
for the intercept
y = data['class']
```

```
#5.8.2. Split the dataset in training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.8.3. Creating a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

#5.8.4. Training the model
clf.fit(X_train, y_train)

#5.8.5. Making predictions
y_pred = clf.predict(X_test)

#5.8.6. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

#5.8.7. Print detailed classification metrics
print(classification_report(y_test, y_pred))

#5.8.8. Feature importances
# Used to retrieve the importance of each feature in the dataset.
# It gives you a score for each feature, the higher the score, the more relevant.
# Measures how much each feature contributes to decreasing the weighted impurity
in a tree
# or in other words how much the model's predictions depend on this feature.
feature_importances = clf.feature_importances_
features = ['u', 'g', 'alpha', 'i']
indices = np.argsort(feature_importances)[::-1]

#5.8.9. Plotting Feature Importances
plt.figure(figsize=(10,6))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), feature_importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), [features[i] for i in indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
Accuracy: 0.8422
              precision    recall  f1-score   support

           0       0.88      0.93      0.91     11860
           1       0.76      0.66      0.71      4343
           2       0.79      0.78      0.79      3797

    accuracy                           0.84     20000
   macro avg       0.81      0.79      0.80     20000
```

```
weighted avg       0.84      0.84      0.84     20000
```




```
#ANALISIS
#The Random Forest Classifier achieved a strong accuracy of 84.22% on the test
set.
#Precision values ranged from 0.76 to 0.88, indicating mostly well accuracies in
predicting each class.
#Recall values varied from 0.66 to 0.93, demonstrating the model's ability to
identify most instances
#of class 0, although in class 1, some instances were missed. The F1-scores,
ranging from 0.71 to 0.91,
#highlight a good balance between precision and recall for all classes. Overall,
the classifier performed
#moderately well across all metrics, showcasing its ability in accurately
classifying instances based on the selected features.
#The bar chart displays the importance of each feature used in the Random Forest
Classifier. \
#i is the most influential feature, with an importance value just below 0.35,
#suggesting it is the main driver for the model's predictions. g and u also shows
considerable importance, although to
#a lesser extent, with values near 0.25. Alpha has the least importance, with a
value slightly lower than
#the others, around 0.15. This implies that while alpha contribute to the model,
they have a smaller
```

```
#impact on the outcome compared to the other features. Integrating this with the
previous analysis, the classifier's
#effectiveness stems from a few key features, with i and g being the most
critical for classification
#decisions.
```

```
#5.9.   Support Vector Machines (TASK 2)

#5.9.1. Create a binary category
data['redshift_level'] = 0 # Create a new column 'redshift_level' initialized
with zeros
data.loc[data['redshift'] > data['redshift'].mean(), 'redshift_level'] = 1  # Set
'redshift_level' to 1 when 'redshift' is above the mean

#5.9.2. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['redshift_level']

#5.9.3. Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#5.9.4. Create a SVM Classifier with a radial basis function kernel
svm_model = SVC(kernel='linear', C=100)  # High C value can lead to overfitting

#5.9.5. Train the model using the training sets
svm_model.fit(X_train, y_train)

#5.9.6. Predict the response for test dataset
y_pred = svm_model.predict(X_test)

#5.9.7. Evaluate the model
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.93      0.86     11860
           1       0.70      0.28      0.40      4343
           2       0.68      0.79      0.73      3797

    accuracy                           0.76     20000
   macro avg       0.73      0.67      0.66     20000
weighted avg       0.75      0.76      0.74     20000
```

```
# ANALYSIS
# The Support Vector Machine (SVM) model with a linear kernel has shown below
performance in classifying the binary
# redshift level category. With precision scores of 0.68 for the lower redshift
level and 0.80 for the higher, coupled with
# recall scores of 0.28 and 0.93, respectively, the model demonstrates average
accuracy in both identifying and predicting the
# correct categories. The F1-scores, which are a harmonic mean of precision and
recall, vary from below average to good, reflecting a
# ok classification performance. An overall accuracy of 75% further confirms the
decency of the model.
# Notably, a macro average precision and recall of 0.73 suggest that the model
performs uniformly well across both categories.
# The high value of C used in the model suggests a strict margin, which could
lead to overfitting, but the current results
# indicate that the model is generalizing ok to the test data.
```

```
#5.10.  K-means - Unsupervised Learning (TASK 1 and TASK 2)

#5.10.1. We need to load some useful functions
# Function for cluster profiling
def cluster_profiling(X, labels, feature_names):
    data = pd.DataFrame(X, columns=feature_names)
    data['Cluster'] = labels
    profile = data.groupby('Cluster').mean()
    return profile

# Function for calculating feature importance
def feature_importance(kmeans, feature_names):
    centroids = kmeans.cluster_centers_
    importance = pd.DataFrame(centroids, columns=feature_names).abs()
    return importance

# Function for cluster validation using silhouette score
def cluster_validation(X, y, labels):
    return silhouette_score(X, labels)

# Anomaly detection function
def anomaly_detection(X, kmeans):
    distances = cdist(X, kmeans.cluster_centers_, 'euclidean')    # Euclidean
distance between each point and all centroids
    distance_to_nearest_centroid = np.min(distances, axis=1)      # Assigning
each point to its nearest centroid
    outlier_threshold = np.percentile(distance_to_nearest_centroid, 95) # Finds
the distance threshold to the centroid for 95% of the data points
    anomaly_indices = np.where(distance_to_nearest_centroid >
outlier_threshold)[0] # Filter in the form of indices
```

```
    return anomaly_indices


#5.10.2. Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i', 'z']]
y = data['class']

#5.10.3. Split the dataset in training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#5.10.4. Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

#5.10.5. Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.labels_

#5.10.6. Cluster Profiling
feature_names = ['u', 'g', 'alpha', 'i', 'z']
profile = cluster_profiling(X_scaled, labels, feature_names)
print("Cluster Profiling: (Predictor mean by clusters)\n", profile)

#5.10.7. Feature Importance
importance = feature_importance(kmeans, feature_names)
print("\nFeature Importance: (Just absolute values)\n", importance)

#5.10.8. Evaluate cluster quality
silhouette_avg = cluster_validation(X_scaled, y_train, labels)
davies_bouldin = davies_bouldin_score(X_scaled, labels)
print("\nCluster Validation Metrics:")
print("Silhouette Score:", silhouette_avg)
print("Davies-Bouldin Score:", davies_bouldin)

#5.10.9. Anomaly Detection
anomaly_indices = anomaly_detection(X_scaled, kmeans)
print("\nAnomaly Detection: (Distance threshold to the centroid for 95% of the
data points)")
print("Indices of Anomalies in the Original Dataset:", anomaly_indices)

# Filter the original DataFrame to get the data of the detected anomalies
anomalies = pd.DataFrame(X, columns=feature_names).iloc[anomaly_indices]
print("\nData of Detected Anomalies:\n", anomalies)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(

Cluster Profiling: (Predictor mean by clusters)
                   u           g     alpha         i           z
Cluster
0           0.031768    0.037033 -0.022039  0.624338    0.033628
1        -282.271051 -282.374240  0.481566 -0.607812 -282.489815
2          -0.047257   -0.056721  0.039618 -1.122751   -0.050593

Feature Importance: (Just absolute values)
              u           g     alpha         i           z
0    0.031795    0.037076  0.020057  0.625261    0.033673
1  282.271051  282.374240  0.481566  0.607812  282.489815
2    0.047164    0.056630  0.035952  1.121296    0.050524

Cluster Validation Metrics:
Silhouette Score: 0.3555037259153444
Davies-Bouldin Score: 0.7426080496266313

Anomaly Detection: (Distance threshold to the centroid for 95% of the data
points)
Indices of Anomalies in the Original Dataset: [   31    37    94 ... 79960 79975
79987]

Data of Detected Anomalies:
                u         g       alpha         i         z
31      23.08039  22.02426  240.213909  19.90149  19.37544
37      20.71552  18.85877  140.512983  17.24668  16.87722
94      23.59629  22.13309    2.170252  19.82567  19.41473
106     21.29609  19.22355  140.679264  17.08743  16.75041
114     23.68024  22.93109  333.110114  21.54711  21.80262
...          ...       ...         ...       ...       ...
79954   22.35834  22.85207  320.019850  21.56027  21.67084
79956   20.38129  18.26966  335.385830  16.93117  16.76495
79960   25.07338  21.77208  135.289804  19.39097  18.96253
79975   22.78397  21.61523  139.004845  19.33000  18.90451
79987   22.90807  23.09484  132.350371  21.16831  20.93390

[4000 rows x 5 columns]
```

```
#ANALYSIS
#In the k-means cluster analysis, three distinct profiles emerged based on the
mean values of u,
#g, alpha, and i. Cluster 0 is defined by below-average alpha levels,
#Cluster 1 has notably lower levels of u, g and z, and Cluster 2 is characterized
by an above-average
```

```
#value of alpha, indicating a higher quality. The silhouette score suggests
moderate
#cluster cohesion and separation, while the Davies-Bouldin score implies
compactness and distinctness among the clusters.
#Anomalies were detected, which diverge significantly from their cluster
centroids, pointing to potential outliers
#with unique properties in the dataset.
```

```
#Step 6. EVALUATION PHASE
#6.1.   Evaluate model performance using k-fold cross-validation.
#6.2.   Evaluate model performance using other related metrics. - DONE in other
sections
#6.3.   Evaluate models for overfitting.
#6.4.   Identify the most important features for each model. - DONE in other
sections
#6.5.   Evaluate classification model performance metrics (accuracy, precision,
recall, and F1 score). - DONE in other sections
#6.6.   Evaluate prediction model performance metrics (MSE (Mean Squared Error),
RMSE (Root Mean Squared Error), and MAE (Mean Absolute Error) for regression).
#6.7.   Report and present the results of your model evaluations, highlighting
strengths and weaknesses.
```

```
#6.1.   Evaluate model performance using k-fold cross-validation.

# 6.1.1. LDA model cross-validation - TASK 1
# a) Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
X = sm.add_constant(X)  # Adds a constant column to input features to account for
the intercept
y = data['class']

# b) Create an LDA model
lda = LinearDiscriminantAnalysis()

# c) Define a 5-fold cross-validation strategy
cv_strategy = KFold(n_splits=5, shuffle=True, random_state=42)

# d) Evaluate model performance using cross-validation
cv_scores = cross_val_score(lda, X, y, cv=cv_strategy, scoring='accuracy')

# e) Print results
print(f"Cross-validation scores for each fold: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")
print(f"Standard deviation of cross-validation scores: {cv_scores.std()}")
```

```
Cross-validation scores for each fold: [0.59875 0.75935 0.60585 0.59825 0.59485]
```

```
Mean cross-validation score: 0.63141
Standard deviation of cross-validation scores: 0.06407004292178989
```

```
# 6.1.2. Decision tree model cross-validation - TASK 1
# a) Selecting a subset of the features
X = data[['u', 'g', 'alpha', 'i']]
y = data['class']

# b) Create a Decision Tree Classifier
classifier = DecisionTreeClassifier(max_depth=10, random_state=42)

# c) Define a 5-fold cross-validation strategy
cv_strategy = KFold(n_splits=5, shuffle=True, random_state=42)

# d) Evaluate model performance using cross-validation
cv_scores = cross_val_score(classifier, X, y, cv=cv_strategy, scoring='accuracy')

# e) Output the results of the cross-validation
print(f"Cross-validation scores for each fold: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")
print(f"Standard deviation of cross-validation scores: {cv_scores.std()}")
```

```
Cross-validation scores for each fold: [0.806  0.8112 0.8111 0.8133 0.8033]
Mean cross-validation score: 0.80898
Standard deviation of cross-validation scores: 0.0037209676160912786
```

```
# ANALYSIS - TASK 1
# The LDA model shows a mean cross-validation accuracy of 0.631 with a standard
deviation of 0.064, suggesting moderate and stable performance.
# In contrast, the Decision Tree Classifier boasts a superior mean accuracy of
0.809 and a lower standard deviation of 0.037,
# indicating a marginally better and more consistent predictive capability. These
results point to the Decision Tree Classifier as a
# marginally more reliable model for this dataset.
# SELECTED MODEL: Decision Tree Classifier
```

```
#Step 7. DEPLOYMENT PHASE
#7.1.   Make predictions and classifications with new data
#       Decision Tree Classifier - TASK 1
#       Logistic Regression - TASK 2
#       Lasso Regression Model - TASK 3

#7.1.1. Manually building df_new based on the sampled data
df_new = pd.DataFrame({
    'u': [154.55, 225.24, 304.60, 205.70, 314.68],  # example values
    'g': [23.03, 21.65, 18.50, 19.25, 19.45],  # example values
```

```
    'alpha': [19.66, 18.25, 20.75, 19.38, 18.04],  # example values
    'i': [20.84, 19.08, 16.80, 19.85, 20.51],  # example values
})
X_new = df_new[['u', 'g', 'alpha', 'i']]

# 7.1.2. Decision Tree Classifier - TASK 1
y_pred_dt = dtc.predict(X_new)
#print("TASK 1 - Decision Tree Predictions:\n", y_pred_dt)

# 7.1.3. Logistic Regression - TASK 2
X_new = sm.add_constant(X_new)
y_pred_lr = model.predict(X_new)
#print("TASK 2 - Logistic Regression Predictions:\n", y_pred_lr)

# 7.1.4. Lasso Regression Model - TASK 3
# Ensure to scale new data as per the trained model's scaling
X_new_scaled = scaler_X.transform(X_new)  # Use the same scaler used during
training
y_pred_lasso = lasso_reg.predict(X_new_scaled)
y_pred_lasso_actual = scaler_y.inverse_transform(y_pred_lasso.reshape(-1,
1)).flatten()  # If the output was scaled
#print("TASK 3 - Lasso Regression Predictions:\n", y_pred_lasso_actual)

# 7.1.5. Printing results
#Header
print(f"{'':<6} {'Decision Tree':<15} {'Logistic Regression':<20} {'Lasso
Regression':<15}")
print(f"{'Index':<6} {'Task 1':<15} {'Task 2':<20} {'Task 3':<15}")

# Rows of the table
for index, (dt, lr, lasso) in enumerate(zip(y_pred_dt, y_pred_lr,
y_pred_lasso_actual)):
    print(f"{index:<6} {dt:<15} {lr:<20.2f} {lasso:<15.3f}")
```

```
       Decision Tree   Logistic Regression  Lasso Regression
Index  Task 1          Task 2               Task 3
0      0               0.00                 0.902
1      0               0.00                 0.576
2      0               0.00                 0.155
3      0               0.00                 0.719
4      1               0.00                 0.841
```