

1.Download a dataset from Kaggle:<https://www.kaggle.com/datasets/joebeachcapital/students-performance?resource=download> This one is a CLASSIFICATION problem

```
import pandas as pd
```

```
df = pd.read_csv('/content/StudentsPerformance_with_headers.csv')
display(df.head())
```

	STUDENT ID	Student Age	Sex	Graduated high-school type	Scholarship type	Additional work	Regular artistic or sports activity	Do you have a partner	Total salary if available	Transportation to the university	... to midterm exams	Preparation F
0	STUDENT1	2	2	3	3	1	2	2	1	1	...	1
1	STUDENT2	2	2	3	3	1	2	2	1	1	...	1
2	STUDENT3	2	2	2	3	2	2	2	2	4	...	1
3	STUDENT4	1	1	1	3	1	2	1	2	1	...	1
4	STUDENT5	2	2	1	3	2	2	1	3	1	...	2

5 rows × 33 columns

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

Mounted at /content/drive

2.Handle missing values appropriately to prepare the datasets for applying machine learning models.

here the code for the type to know how we can work which type of data numerical or categorical

Start coding or [generate](#) with AI.

```
display(df.dtypes)
```

	0
STUDENT ID	object
Student Age	int64
Sex	int64
Graduated high-school type	int64
Scholarship type	int64
Additional work	int64
Regular artistic or sports activity	int64
Do you have a partner	int64
Total salary if available	int64
Transportation to the university	int64
Accommodation type in Cyprus	int64
Mother's education	int64
Father's education	int64
Number of sisters/brothers	int64
Parental status	int64
Mother's occupation	int64
Father's occupation	int64
Weekly study hours	int64
Reading frequency	int64
Reading frequency.1	int64
Attendance to the seminars/conferences related to the department	int64
Impact of your projects/activities on your success	int64
Attendance to classes	int64
Preparation to midterm exams 1	int64
Preparation to midterm exams 2	int64
Taking notes in classes	int64
Listening in classes	int64
Discussion improves my interest and success in the course	int64
Flip-classroom	int64
Cumulative grade point average in the last semester (/4.00)	int64
Expected Cumulative grade point average in the graduation (/4.00)	int64
COURSE ID	int64
GRADE	int64

dtype: object

handling missing data of there is a missing (no missing data here = 0)

```
df.isnull().sum()
```

	0
STUDENT ID	0
Student Age	0
Sex	0
Graduated high-school type	0
Scholarship type	0
Additional work	0
Regular artistic or sports activity	0
Do you have a partner	0
Total salary if available	0
Transportation to the university	0
Accommodation type in Cyprus	0
Mother's education	0
Father's education	0
Number of sisters/brothers	0
Parental status	0
Mother's occupation	0
Father's occupation	0
Weekly study hours	0
Reading frequency	0
Reading frequency.1	0
Attendance to the seminars/conferences related to the department	0
Impact of your projects/activities on your success	0
Attendance to classes	0
Preparation to midterm exams 1	0
Preparation to midterm exams 2	0
Taking notes in classes	0
Listening in classes	0
Discussion improves my interest and success in the course	0
Flip-classroom	0
Cumulative grade point average in the last semester (/4.00)	0
Expected Cumulative grade point average in the graduation (/4.00)	0
COURSE ID	0
GRADE	0

dtype: int64

import Z-scores to Handle outliersStart coding or [generate](#) with AI.

```

from scipy.stats import zscore
import numpy as np

# Identify numerical columns for outlier detection
numerical_cols = df.select_dtypes(include=np.number).columns

# Calculate Z-scores for numerical columns
z_scores = np.abs(df[numerical_cols].apply(zscore))

# Define a threshold for outliers (e.g., 3 standard deviations)
threshold = 3

```

```
# Identify outliers
outlier_mask = (z_scores > threshold).any(axis=1)

print(f"Number of outliers detected: {df[outlier_mask].shape[0]}")

# Display some rows with outliers (optional)
display(df[outlier_mask].head())

# Example of handling outliers: Replace with the mean of the column (you might choose median or capping as well)
df_no_outliers = df.copy()
for col in numerical_cols:
    col_z_scores = np.abs(zscore(df_no_outliers[col]))
    outliers_in_col = df_no_outliers[col_z_scores > threshold]
    if not outliers_in_col.empty:
        mean_val = df_no_outliers[col].mean()
        df_no_outliers.loc[col_z_scores > threshold, col] = mean_val

print("\nDataFrame after replacing outliers with column mean (first 5 rows of original df):")
display(df_no_outliers.head())
```

Number of outliers detected: 28

	STUDENT ID	Student Age	Sex	Graduated high-school type	Scholarship type	Additional work	Regular artistic or sports activity	Do you have a partner	Total salary if available	Transportation to the university	...	Preparation to midterm exams 1
11	STUDENT12	1	1	1	4	1	1	2	4	2	...	3
15	STUDENT16	2	2	2	3	2	2	2	1	1	...	1
18	STUDENT19	1	1	2	4	2	2	2	3	1	...	1
23	STUDENT24	3	2	2	2	1	1	2	5	1	...	1
47	STUDENT48	2	2	2	3	2	2	1	1	1	...	1

5 rows × 33 columns

DataFrame after replacing outliers with column mean (first 5 rows of original df):

```
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
/tmp/ipython-input-792276010.py:28: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in the future.
df_no_outliers.loc[col_z_scores > threshold, col] = mean_val
```

	STUDENT ID	Student Age	Sex	Graduated high-school type	Scholarship type	Additional work	Regular artistic or sports activity	Do you have a partner	Total salary if available	Transportation to the university	...	Preparation to midterm exams 1
0	STUDENT1	2	2	3	3.0	1	2	2	1.0	1	...	1
1	STUDENT2	2	2	3	3.0	1	2	2	1.0	1	...	1
2	STUDENT3	2	2	2	3.0	2	2	2	2.0	4	...	1
3	STUDENT4	1	1	1	3.0	1	2	1	2.0	1	...	1
4	STUDENT5	2	2	1	3.0	2	2	1	3.0	1	...	2

5 rows × 33 columns

```
display(df_no_outliers)
```

arship type	Additional work	Regular artistic or sports activity	Do you have a partner	Total salary if available	Transportation to the university	...	Preparation to midterms exams
3.0	1	2	2	1.000000	1	...	
3.0	1	2	2	1.000000	1	...	
3.0	2	2	2	2.000000	4	...	
3.0	1	2	1	2.000000	1	...	
3.0	2	2	1	3.000000	1	...	
...	...	...	...	...	...	...	
3.0	1	1	2	1.000000	1	...	
4.0	2	2	2	1.000000	4	...	
4.0	2	2	2	1.000000	1	...	
4.0	1	1	1	1.627586	2	...	
5.0	2	2	2	3.000000	1	...	

3. Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.

Define Target (y) and Features (X)

```
X = df_no_outliers.drop('GRADE', axis=1)
y = df_no_outliers['GRADE']
```

```
X = X.drop('STUDENT ID', axis=1)
X_processed = pd.get_dummies(X, columns=categorical_cols_to_encode, drop_first=True)

display(X_processed.head())
```

	Student Age_2	Student Age_3	Sex_2	Graduated high- school type_2	Graduated high- school type_3	Scholarship type_3.0	Scholarship type_3.5724137931034483	Scholarship type_4.0	Scholarship type_5.0	Additional work_2	...
0	True	False	True	False	True	True	False	False	False	False	...
1	True	False	True	False	True	True	False	False	False	False	...
2	True	False	True	True	False	True	False	False	False	True	...
3	False	False	False	False	False	True	False	False	False	False	...
4	True	False	True	False	False	True	False	False	False	True	...

5 rows × 85 columns

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

for col in X.columns:
    X[col] = encoder.fit_transform(X[col])
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

****Apply ML Models with Grid Search and they are 3 type : 1. Logistic Regression 2.Decision Tree 3.Random Forest 4.KNN 5.Nighf based ,But I used only the first 3 modles ** bold text**

Model 1: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

log_reg = LogisticRegression(max_iter=1000)

param_grid_lr = {
    'C': [0.01, 0.1, 1, 10]
}

grid_lr = GridSearchCV(log_reg, param_grid_lr, cv=5, scoring='accuracy')
grid_lr.fit(X_train, y_train)

best_lr = grid_lr.best_estimator_
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The least populated class in y has c
warnings.warn(
```

Model 2: Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)

param_grid_dt = {
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='accuracy')
grid_dt.fit(X_train, y_train)

best_dt = grid_dt.best_estimator_
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The least populated class in y has c
warnings.warn(
```

Model 3: Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)

param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20]
}

grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='accuracy')
grid_rf.fit(X_train, y_train)

best_rf = grid_rf.best_estimator_
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The least populated class in y has c
warnings.warn(
```

IAST STEP Evaluate Models

```
from sklearn.metrics import accuracy_score, classification_report
```

```

models = {
    "Logistic Regression": best_lr,
    "Decision Tree": best_dt,
    "Random Forest": best_rf
}

for name, model in models.items():
    y_pred = model.predict(X_test)
    print("=====")
    print(f"Model: {name}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

```

```

=====
Model: Logistic Regression
Accuracy: 0.20689655172413793
Classification Report:

```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.10	0.33	0.15	3
2	0.20	0.25	0.22	4
3	0.33	0.33	0.33	6
4	0.00	0.00	0.00	3
5	0.25	0.33	0.29	3
6	1.00	0.25	0.40	4
7	0.00	0.00	0.00	2
accuracy			0.21	29
macro avg	0.24	0.19	0.17	29
weighted avg	0.27	0.21	0.20	29

```

=====
Model: Decision Tree
Accuracy: 0.1724137931034483
Classification Report:

```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.17	0.67	0.27	3
2	0.33	0.25	0.29	4
3	0.25	0.17	0.20	6
4	0.00	0.00	0.00	3
5	0.20	0.33	0.25	3
6	0.00	0.00	0.00	4
7	0.00	0.00	0.00	2
accuracy			0.17	29
macro avg	0.12	0.18	0.13	29
weighted avg	0.14	0.17	0.13	29

```

=====
Model: Random Forest
Accuracy: 0.1724137931034483
Classification Report:

```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.12	0.67	0.20	3
2	0.33	0.25	0.29	4
3	0.33	0.33	0.33	6
4	0.00	0.00	0.00	3
5	0.00	0.00	0.00	3
6	0.00	0.00	0.00	4
7	0.00	0.00	0.00	2
accuracy			0.17	29
macro avg	0.10	0.16	0.10	29
weighted avg	0.13	0.17	0.13	29

****4. Use two feature selection methods to identify the most important features from the datasets. I'll use from filter methods: 1 Chi-Square Test 2 ANOVA ****

```

from sklearn.feature_selection import SelectKBest, chi2

chi2_selector = SelectKBest(score_func=chi2, k=5)
X_chi2 = chi2_selector.fit_transform(X, y)

```

```
chi2_features = X.columns[chi2_selector.get_support()]

print("Top features selected using Chi-Square:")
print(chi2_features)
```

```
Top features selected using Chi-Square:
Index(['Total salary if available', 'Mother's education',
      'Impact of your projects/activities on your success',
      'Cumulative grade point average in the last semester (/4.00)',
      'COURSE ID'],
      dtype='object')
```

```
from sklearn.feature_selection import SelectKBest, f_classif

anova_selector = SelectKBest(score_func=f_classif, k=5)
X_anova = anova_selector.fit_transform(X, y)

anova_features = X.columns[anova_selector.get_support()]

print("\nTop features selected using ANOVA F-test:")
print(anova_features)
```

```
Top features selected using ANOVA F-test:
Index(['Sex', 'Impact of your projects/activities on your success',
      'Cumulative grade point average in the last semester (/4.00)',
      'Expected Cumulative grade point average in the graduation (/4.00)',
      'COURSE ID'],
      dtype='object')
```

Here I'll use RFE from Warpped Method

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)

rfe = RFE(estimator=model, n_features_to_select=5)
X_rfe = rfe.fit_transform(X, y)

rfe_features = X.columns[rfe.support_]

print("Top features selected using RFE:")
print(rfe_features)
```

```
Top features selected using RFE:
Index(['Student Age', 'Sex', 'Do you have a partner',
      'Impact of your projects/activities on your success',
      'Taking notes in classes'],
      dtype='object')
```

****5.**Apply machine learning models to the selected features and evaluate the models.

✓ 1.Prepare datasets with selected features

Chi-Square features

2.RFE feature

3.Apply ML models

4.Evaluate models using Chi-Square features

5.Evaluate models using RFE features**

```
X_chi2_df = pd.DataFrame(X_chi2, columns=chi2_features)

Xc_train, Xc_test, yc_train, yc_test = train_test_split(
```

```
X_chi2_df, y, test_size=0.2, random_state=42
)
```

```
X_rfe_df = pd.DataFrame(X_rfe, columns=rfe_features)

Xr_train, Xr_test, yr_train, yr_test = train_test_split(
    X_rfe_df, y, test_size=0.2, random_state=42
)
```

```
evaluate_models(
    Xc_train, Xc_test, yc_train, yc_test,
    "Evaluation using Chi-Square Selected Features"
)
```

```
=====
```

```
Model: Logistic Regression
Accuracy: 0.1724137931034483
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.09	0.33	0.14	3
2	0.14	0.25	0.18	4
3	0.67	0.33	0.44	6
4	0.00	0.00	0.00	3
5	0.33	0.33	0.33	3
6	0.00	0.00	0.00	4
7	0.00	0.00	0.00	2
accuracy				29
macro avg	0.15	0.16	0.14	29
weighted avg	0.20	0.17	0.17	29

```
Model: Decision Tree
Accuracy: 0.4827586206896552
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.20	1.00	0.33	3
2	0.33	0.25	0.29	4
3	1.00	0.33	0.50	6
4	1.00	0.33	0.50	3
5	1.00	0.33	0.50	3
6	1.00	1.00	1.00	4
7	0.67	1.00	0.80	2
accuracy				29
macro avg	0.65	0.53	0.49	29
weighted avg	0.66	0.48	0.47	29

```
Model: Random Forest
Accuracy: 0.27586206896551724
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.14	0.33	0.20	3
2	0.29	0.50	0.36	4
3	0.50	0.33	0.40	6
4	0.00	0.00	0.00	3
5	0.00	0.00	0.00	3
6	0.50	0.25	0.33	4
7	0.33	1.00	0.50	2
accuracy				29
macro avg	0.22	0.30	0.22	29
weighted avg	0.25	0.28	0.23	29

```
evaluate_models(
    Xr_train, Xr_test, yr_train, yr_test,
    "Evaluation using RFE Selected Features"
)
```

```
=====
```

```
Evaluation using RFE Selected Features
```

```

=====
Model: Logistic Regression
Accuracy: 0.13793103448275862
Classification Report:

```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.12	1.00	0.21	3
2	0.25	0.25	0.25	4
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	3
5	0.00	0.00	0.00	3
6	0.00	0.00	0.00	4
7	0.00	0.00	0.00	2
accuracy			0.14	29
macro avg	0.05	0.16	0.06	29
weighted avg	0.05	0.14	0.06	29

```

Model: Decision Tree
Accuracy: 0.13793103448275862
Classification Report:

```

	precision	recall	f1-score	support
0	0.50	0.25	0.33	4
1	0.06	0.33	0.10	3
2	0.20	0.25	0.22	4
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	3
5	0.00	0.00	0.00	3
6	0.00	0.00	0.00	4
7	0.25	0.50	0.33	2
accuracy			0.14	29
macro avg	0.13	0.17	0.12	29
weighted avg	0.12	0.14	0.11	29

```

Model: Random Forest
Accuracy: 0.20689655172413793
Classification Report:

```

	precision	recall	f1-score	support
0	1.00	0.25	0.40	4
1	0.09	0.33	0.14	3
2	0.33	0.50	0.40	4
3	0.40	0.33	0.36	6
4	0.00	0.00	0.00	3
5	0.00	0.00	0.00	3
6	0.00	0.00	0.00	4
7	0.00	0.00	0.00	2
accuracy			0.21	29

Visualization Options Bar char

```

import pandas as pd

results = []

for name, model in models.items():
    model.fit(Xc_train, yc_train)
    y_pred = model.predict(Xc_test)
    acc = accuracy_score(yc_test, y_pred)
    results.append([name, acc])

results_df = pd.DataFrame(results, columns=["Model", "Accuracy"])
print(results_df)

```

	Model	Accuracy
0	Logistic Regression	0.172414
1	Decision Tree	0.482759
2	Random Forest	0.275862

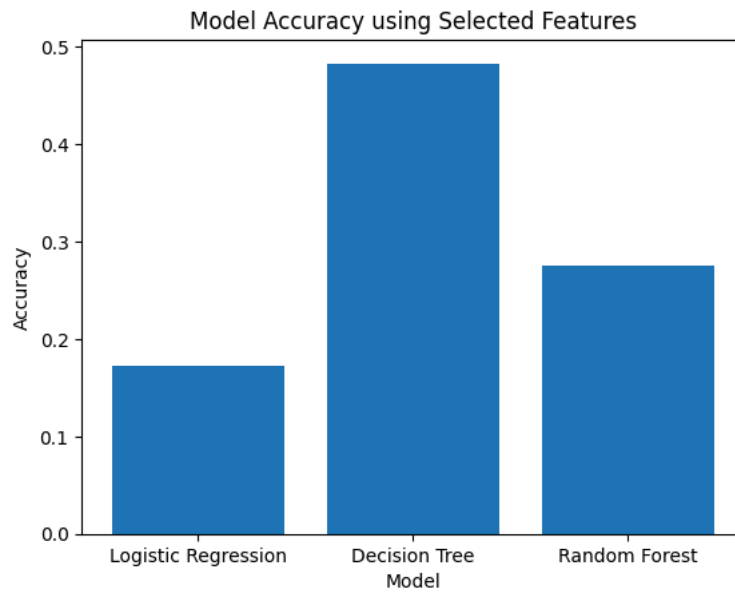
```

import matplotlib.pyplot as plt

plt.figure()
plt.bar(results_df["Model"], results_df["Accuracy"])
plt.title("Model Accuracy using Selected Features")

```

```
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.show()
```



Matrix Visualization

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

rf_model = models["Random Forest"]
rf_model.fit(Xc_train, yc_train)
y_pred_rf = rf_model.predict(Xc_test)

cm = confusion_matrix(yc_test, y_pred_rf)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

