INTRODUCTION TO DATA SCIENCE
MILESTONE – II

Title

Student Job and Course Recommendation System

Introduction

Students often struggle to find the right career path, job opportunities, or skill-enhancing courses that align with their aspirations. This project aims to bridge that gap using data-driven insights. By analyzing student profiles, we provide personalized recommendations to help students make informed career decisions by suggesting to them an online course and available job roles.

Our approach involves Exploratory Data Analysis (EDA) to understand key patterns in student academic backgrounds, career goals, and skillsets. We then implement Machine Learning models to recommend the most relevant job postings and courses tailored to each student's profile. This system empowers students by guiding them toward the right opportunities, ultimately improving their career prospects.

Objective

Choosing the right career path can be overwhelming for students. This project aims to simplify that journey by providing personalized job and course recommendations based on their academic background, skills, and career goals.

- Helps students find the right job opportunities by matching their skills and interests with relevant job roles.
- Suggests online courses that can help them develop in-demand skills and improve their chances of landing a great job.

Tool and Technology Stack

Programming Languages: Python, JavaScript

Frameworks & Tools:

- Flask: For building web applications.
- HTML, CSS, JavaScript, Streamlit – For front-end development and user interaction.
- Visual Studio Code – The development environment.

Libraries & Data Analysis:

- Pandas – For data manipulation and preprocessing.
- Matplotlib, Seaborn – For data visualization and insights.
- Scikit-learn – For implementing machine learning models.

Machine Learning Models:

- Support Vector Classifier (SVC)
- Decision Tree Classifier
- Random Forest Classifier

Data Used

This project utilizes three key datasets to provide job and course recommendations.

Datasets:

- **Studentdata.csv** – Contains student academic and career-related information.
- **Online_Courses.csv** – Lists various online courses with relevant details.
- **Job_Postings.csv** – Includes job opportunities along with required skills and company details.

Key Features:

1. Student Data

- **Branch** – The student's field of study.
- **Percentage_10th** – Academic performance till 10th grade.
- **Percentage_12th** – Academic performance in high school.
- **Career_Goal** – The student's desired career path.
- **Skills** – Skills that the student has.

2. Courses Data

- **Title** – Name of the course.
- **Category** – The field or domain of the course.
- **Course Type** – The mode of delivery (Course, Specialization, etc.).

3. Job Data

- **Job Title** – The role title.
- **Job Skills** – The skills required for the position.

Data Source & Structure:

- **Source:** Kaggle (public datasets).
- **Format:** CSV files.

Project Timeline

Task	Date
Task 1: Model Hyperparameter Tuning	04/06/2025 - 04/11/2025
Task 2: Website Development	04/12/2025 - 04/17/2025
Task 3: Integration of Models with the website	04/18/2025 - 04/21/2025
Task 4: Final Report and Submission	04/22/2025 - 05/23/2025

EDA RECAP

Student Data Insights

- High School Percentage Distribution: Most students have scores between 75-95%, showing a strong academic background.
- Career Goal Trends: The most common aspirations include Machine Learning Engineer, Web Developer, and Data Scientist roles.
- Top Student Skills: Popular skills include Python, Java, SQL, HTML, CSS, JavaScript, Machine Learning, and Deep Learning.

Course Data Insights

- Type of Courses Offered: Standard courses are the most preferred, surpassing specializations, professional certificates, and project-based courses.
- Top Skills in Courses: Courses frequently focus on Data Analysis, Python Programming, Data Science, and Machine Learning.
- Top Course Categories: Business and Data Science dominate, while Social Science and Language Learning have fewer offerings.

Job Market Landscape

- Job Titles with Highest Demand: The most frequently posted job roles include Software Engineer, Data Engineer, Business Analyst, and Data Scientist.
- Job Locations: The United States leads in job postings, with California, New York, Chicago, and Texas as major hubs.
- Top Hiring Companies: Companies like Toptal, Jobot, and Perficient are among the top recruiters.
- Skill Requirements: A clear skill gap exists between student profiles and market demands, emphasizing the need for targeted upskilling.

Conclusion

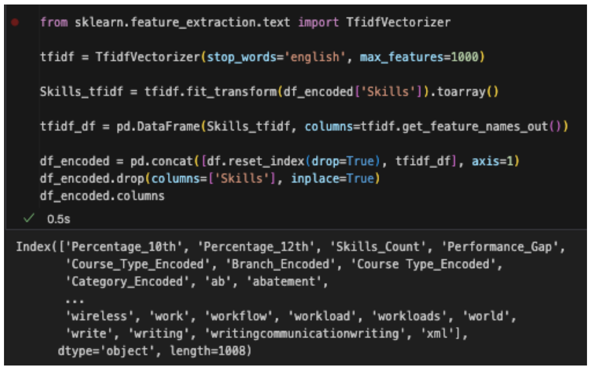
The EDA phase has provided crucial insights into student skills, learning preferences, and job market trends. These findings will shape our recommendation system, ensuring accurate, personalized career and course suggestions that bridge the gap between education and industry. These insights will give us an idea about the data distribution which will help our model selection, feature engineering, and training approaches in the next project phase.

Feature Engineering - Feature Creation

Four key features were engineered to enhance the dataset: Skills Vector, Skills Count, Performance Gap, and Academic Performance Score.

Skills Vector (TF-IDF Vectorization)

- **Purpose**: Converts the unstructured "Skills" text into a numerical representation for better matching with jobs and courses.
- **Impact**: Helps in finding similarities between students' skills and job/course requirements.



```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(stop_words='english', max_features=1000)

Skills_tfidf = tfidf.fit_transform(df_encoded['Skills']).toarray()

tfidf_df = pd.DataFrame(Skills_tfidf, columns=tfidf.get_feature_names_out())

df_encoded = pd.concat([df.reset_index(drop=True), tfidf_df], axis=1)
df_encoded.drop(columns=['Skills'], inplace=True)
df_encoded.columns
```

```
✓ 0.5s
Index(['Percentage_10th', 'Percentage_12th', 'Skills_Count', 'Performance_Gap',
       'Course_Type_Encoded', 'Branch_Encoded', 'Course Type_Encoded',
       'Category_Encoded', 'ab', 'abatement',
       ...
       'wireless', 'work', 'workflow', 'workload', 'workloads', 'world',
       'write', 'writing', 'writingcommunicationwriting', 'xml'],
      dtype='object', length=1008)
```

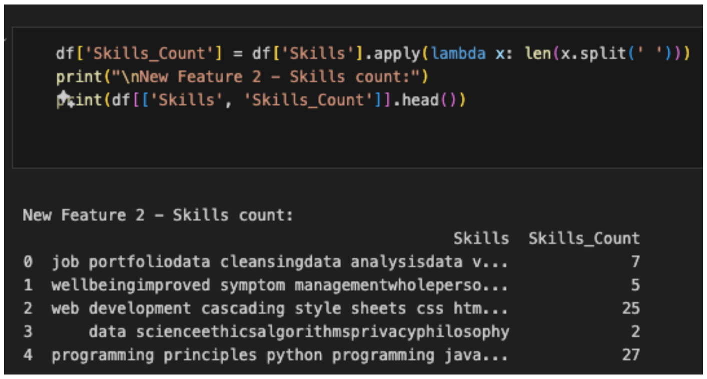
Fig. 1. Feature extraction using TF-IDF

Output Explanation

Skills column is extracted and vectorized using TD-IDF in such a way that multiple new columns are created like wireless, work, workflow, etc. These are fed to the model to improve the chance of a better model performance.

Skills Count

- **Purpose**: Provides a numerical representation of how many distinct skills a course or job requires.
- **Impact**: Helps filter recommendations based on skill intensity (e.g., beginner-friendly courses may have fewer skills listed).



```
df['Skills_Count'] = df['Skills'].apply(lambda x: len(x.split(' ')))
print("\nNew Feature 2 - Skills count:")
print(df[['Skills', 'Skills_Count']].head())
```

```
New Feature 2 - Skills count:
                                              Skills  Skills_Count
0  job portfoliodata cleansingdata analysisdata v...             7
1  wellbeingimproved symptom managementwholeperso...             5
2  web development cascading style sheets css htm...            25
3      data scienceethicsalgorithmsprivacyphilosophy             2
4  programming principles python programming java...            27
```

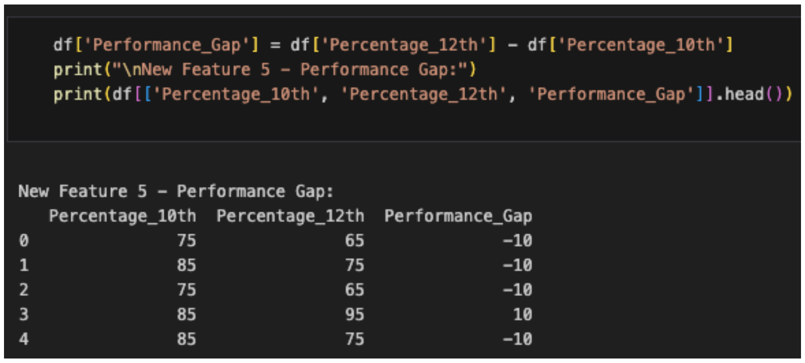
Fig. 2. Skills count declaration

Output Explanation

The number of specific keywords in the skills are extracted and their count is stored to be fed to the machine learning model.

Performance Gap

- **Purpose**: Measures academic improvement or decline from 10th to 12th grade.
- **Impact**: Could indicate learning ability and discipline, which might be relevant for job or course recommendations.



```
df['Performance_Gap'] = df['Percentage_12th'] - df['Percentage_10th']
print("\nNew Feature 5 - Performance Gap:")
print(df[['Percentage_10th', 'Percentage_12th', 'Performance_Gap']].head())
```

```
New Feature 5 - Performance Gap:
   Percentage_10th  Percentage_12th  Performance_Gap
0               75               65              -10
1               85               75              -10
2               75               65              -10
3               85               95               10
4               85               75              -10
```

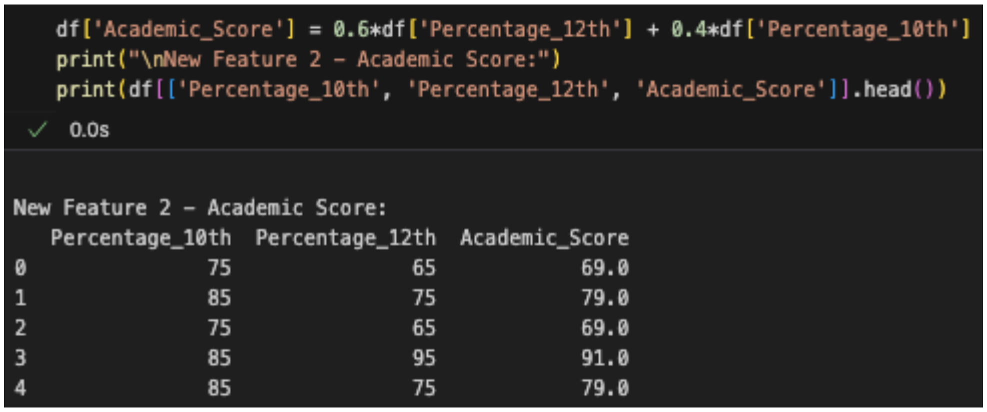
Fig. 3. Performance gap declaration

Output Explanation

The output explains that if there is a negative integer in Performance Gap, then it means that there is a drop in the performance in 12th compared to 10th.

Academic Performance Score

- **Purpose**: A weighted combination of 10th and 12th-grade scores to create a holistic academic metric.
- **Impact**: Helps rank students in terms of academic strength, allowing better job/course filtering.



```
df['Academic_Score'] = 0.6*df['Percentage_12th'] + 0.4*df['Percentage_10th']
print("\nNew Feature 2 - Academic Score:")
print(df[['Percentage_10th', 'Percentage_12th', 'Academic_Score']].head())
✓ 0.0s
```

```
New Feature 2 - Academic Score:
   Percentage_10th  Percentage_12th  Academic_Score
0               75               65            69.0
1               85               75            79.0
2               75               65            69.0
3               85               95            91.0
4               85               75            79.0
```

Fig. 4. Academic score declaration

Output Explanation

The Academic score is derived by multiplying 0.6 with the 12th percentage and 0.4 with the 10th percentage. This gives a better idea of the students' academic performance.

Feature Inclusion Explanation

To make course and job recommendations more accurate, four important features were added to the dataset: Skills Vector, Skills Count, Performance Gap, and Academic Performance Score. The Skills Vector, created using TF-IDF, turns skill descriptions into numbers, making it easier to match students with relevant courses and jobs. Skills Count simply measures how many skills a course or job requires, helping to differentiate between beginner-friendly and advanced opportunities. The Performance Gap looks at how a student's grades changed from 10th to 12th grade, which can indicate improvement, consistency, or a drop in performance which is useful for tailoring recommendations. Finally, the Academic Performance Score combines these grades into a single metric, giving a clearer picture of a student's overall academic ability. Together, these features make the recommendation system more personalized and effective, ensuring students get suggestions that truly fit their skills and career goals

Feature Engineering - Categorical Variable Encoding

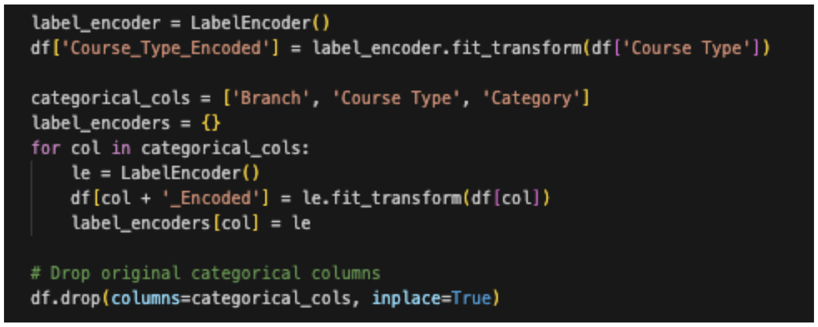
Overview of Categorical Encoding

The dataset contains several categorical variables that need to be transformed into numerical format for machine learning algorithms. In this phase, Label Encoding was applied to convert categorical variables into numerical representations.

Encoding Process

The following categorical variables were encoded:

- Branch (student's academic branch/major)
- Course Type (Course or Specialization)
- Category (subject category of the course)



```python
label_encoder = LabelEncoder()
df['Course_Type_Encoded'] = label_encoder.fit_transform(df['Course Type'])

categorical_cols = ['Branch', 'Course Type', 'Category']
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col + '_Encoded'] = le.fit_transform(df[col])
    label_encoders[col] = le

# Drop original categorical columns
df.drop(columns=categorical_cols, inplace=True)
```

Fig. 5. Encoding using Label Encoder

Reason for choosing Label Encoder

Categorical variables are encoded using Label Encoding for efficiency and model compatibility. This method is ideal as it keeps the dataset compact, works well with tree-based models, and preserves ordinal relationships if present. It avoids the dimensionality explosion of one-hot encoding, making it a practical choice for handling Branch, Course Type, and Category.

Encoding Results

- **Transformation Examples**

Original Category	Encoded Value
Course	0
Specialization	3

- **Branch Encoding**

  The Branch variable (e.g., MCA, CSIT, CSME) was encoded with values: MCA → 7, CSIT → 2, CSME → 3.

- **Category Encoding**

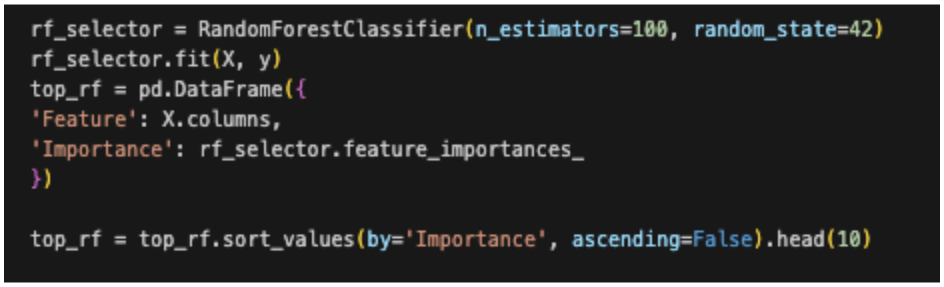
  The Category variable was encoded with values: Data Science → 5, Health → 6, Computer Science → 4.

Feature Selection - Feature Importance Evaluation

Dataset Information (Dependent and Independent variables)

- Features matrix (X): (3010, 1007)
- Target variable (y): (3010,)

1. **Random Forest Feature Importance:** The Random Forest Classifier identified the following top 10 features based on their importance scores.



```
rf_selector = RandomForestClassifier(n_estimators=100, random_state=42)
rf_selector.fit(X, y)
top_rf = pd.DataFrame({
'Feature': X.columns,
'Importance': rf_selector.feature_importances_
})

top_rf = top_rf.sort_values(by='Importance', ascending=False).head(10)
```

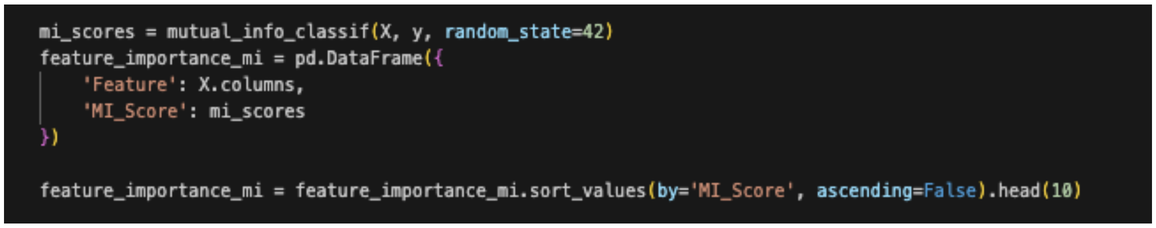
Fig. 6. Feature importance evaluation using RF

RF Feature Importance Table

Feature	Importance
Course_Type_Encoded	0.300561
Skills_Count	0.115456
machine	0.027480
science	0.026603
programming	0.022490
sql	0.020471
analysis	0.020210
data	0.019190
ai	0.010216
learning	0.010078

Table. 1. Feature importance evaluation using RF

2. **Mutual Information Feature Importance:** The Mutual Information Classifier identified the following top 10 features based on their MI scores.



```
mi_scores = mutual_info_classif(X, y, random_state=42)
feature_importance_mi = pd.DataFrame({
    'Feature': X.columns,
    'MI_Score': mi_scores
})

feature_importance_mi = feature_importance_mi.sort_values(by='MI_Score', ascending=False).head(10)
```

Fig. 7. Feature importance evaluation using MI

MI Score Table

Feature	MI Score

Course_Type_Encoded	0.837254
Skills_Count	0.488188
data	0.315966
analysis	0.171076
programming	0.156920
learning	0.137947
machine	0.131653
visualization	0.112272
dataviz	0.099170
science	0.093401

Table. 2. Feature importance evaluation using MI

Feature Importance Plot

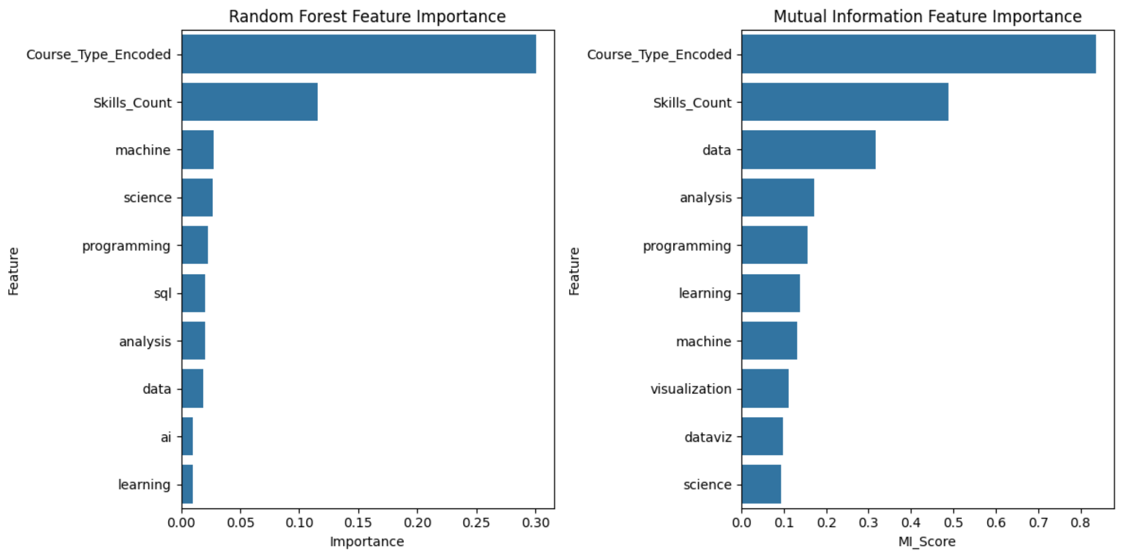
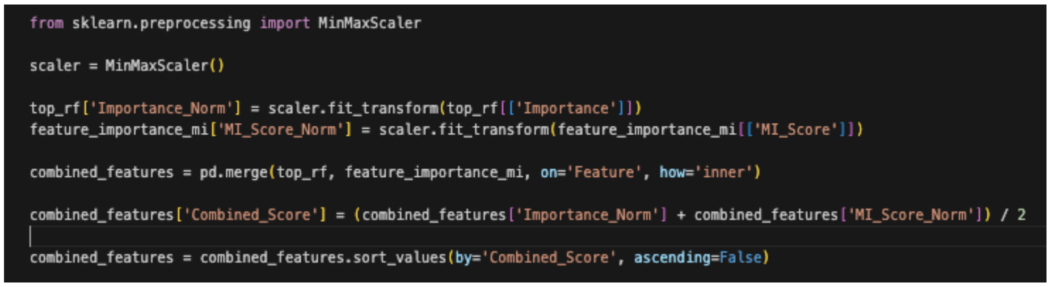


Fig. 8. Comparison between RF and MI

Combining feature importance methodologies

We used MinMaxScaler() to scale the values and then we combined both the methodologies by using the average of both the values for every specific feature.



```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

top_rf['Importance_Norm'] = scaler.fit_transform(top_rf[['Importance']])
feature_importance_mi['MI_Score_Norm'] = scaler.fit_transform(feature_importance_mi[['MI_Score']])

combined_features = pd.merge(top_rf, feature_importance_mi, on='Feature', how='inner')

combined_features['Combined_Score'] = (combined_features['Importance_Norm'] + combined_features['MI_Score_Norm']) / 2

combined_features = combined_features.sort_values(by='Combined_Score', ascending=False)
```

Fig. 9. Merging RF and MI

Combined Feature Importance Plot

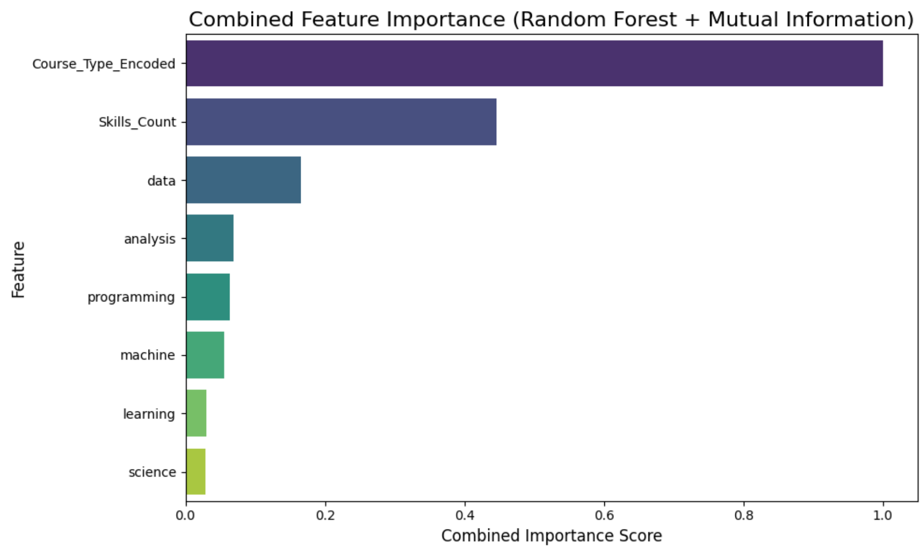


Fig. 10. Merged feature importance

Key Observations

- **Course Type Dominance**: Course_Type_Encoded consistently ranks as the most important feature in both Random Forest and Mutual Information methodologies, highlighting its strong influence on recommendations.

- **Skills Matter:** Skills_Count ranks second in both methods, reinforcing the idea that the number of skills associated with a course or job significantly impacts matching.

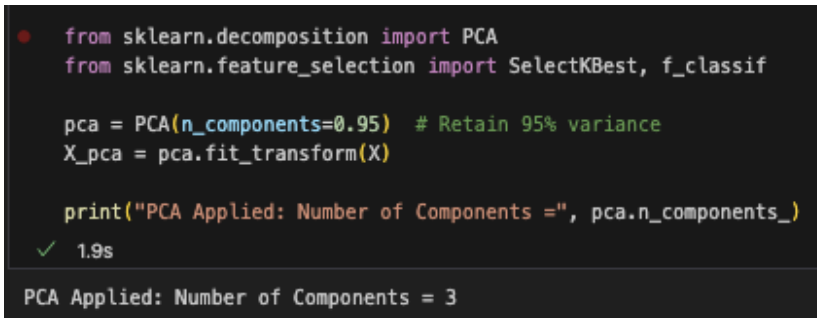
- **Technical Keywords Are Influential:** Features like machine, programming, sql, analysis, data, and science appear in both rankings, indicating that courses with these keywords are more relevant for recommendations.

- **Mutual Information Highlights Domain-Specific Features:** MI scores give higher weight to domain-specific terms like visualization and dataviz. This does not appear in RF rankings; this suggests that they might provide more nuanced insights into skill relevance.

- **Balanced Feature Selection:** The combined feature importance methodology (averaging RF and MI scores) ensures a comprehensive ranking, balancing statistical dependencies (MI) with model-driven relevance (RF). This approach refines recommendations by capturing both predictive power and feature relationships.

Feature Selection - Feature Selection/Dimensionality Reduction

PCA Analysis



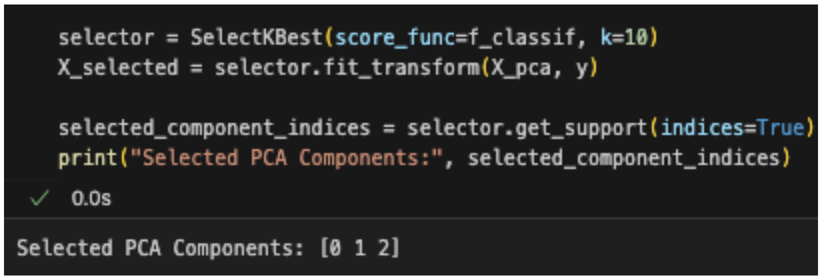
```
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif

pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X)

print("PCA Applied: Number of Components =", pca.n_components_)
```

```
PCA Applied: Number of Components = 3
```

Fig. 11. PCA dimensionality reduction

- **Method**: Principal Component Analysis with variance retention of 95%
- **Result**: The dimensionality was reduced from 1,007 features to just **3 principal components**
- **Significance**: This represents a 99.7% reduction in dimensionality while preserving 95% of the original variance

Feature Selection on PCA Components



```
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X_pca, y)

selected_component_indices = selector.get_support(indices=True)
print("Selected PCA Components:", selected_component_indices)
```

```
✓ 0.0s
```

```
Selected PCA Components: [0 1 2]
```

Fig. 12. Feature selection using PCA

- **Method**: SelectKBest with ANOVA F-value scoring (f_classif)
- **Parameter**: k=10 (maximum of 10 components requested)
- **Result**: All 3 PCA components were selected as significant
- **Selected Components**: [0, 1, 2]

Data Preparation for Modeling

1. **Train-Test Split**:
    a. Test size: 15% of the dataset
    b. Random state: 42
    c. Stratification: Applied to maintain class distribution in both sets
2. **Class Balancing**:
    a. Method: SMOTE (Synthetic Minority Over-sampling Technique)
    b. Applied only to the training set to eliminate class imbalance problem.
    c. Random state: 42

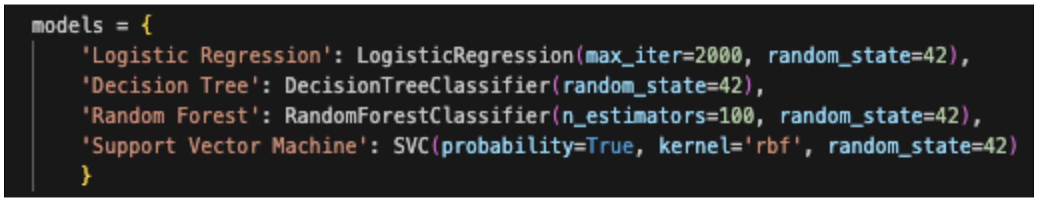
Key Insights

- **Extreme Dimensionality Reduction**: The reduction from 1,007 features to 3 features indicates that most of the original features contained redundant or non-essential information.
- **All Components Selected**: The fact that SelectKBest retained all 3 PCA components suggests that each component contributes significantly to predicting the target variable.
- **Efficient Information Preservation**: The workflow successfully distilled the essential predictive information into just 3 features, which should lead to:
    a. Faster model training
    b. Reduced risk of overfitting

This dimensionality reduction approach has successfully reduced the complexity of the data while maintaining most of its informational content, creating an excellent foundation for efficient model building.

Data Modeling - Model Training, Selection, and Evaluation

Models Used

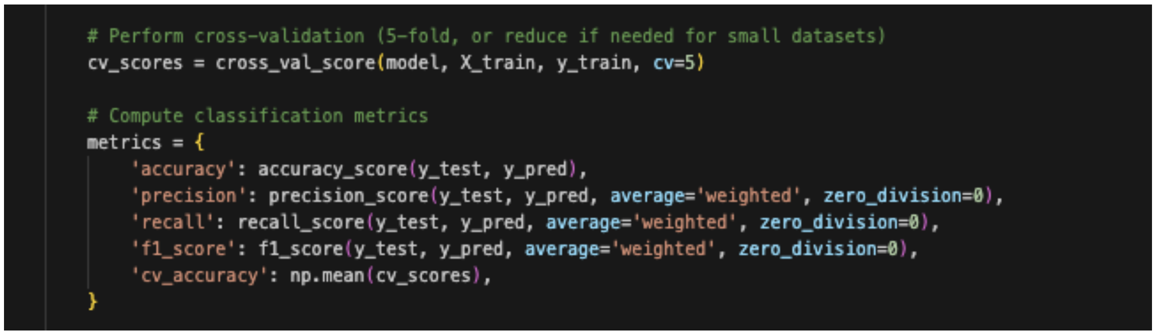


```python
models = {
    'Logistic Regression': LogisticRegression(max_iter=2000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Support Vector Machine': SVC(probability=True, kernel='rbf', random_state=42)
    }
```

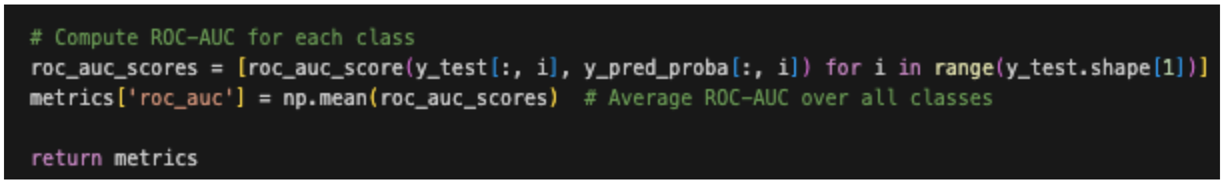
Fig. 13. Model declaration

1. **Logistic Regression** – A simple, fast model that works well when data is linearly separable. It provides probability scores, making it easy to interpret.
2. **Decision Tree** – Splits data into decision paths, making it easy to understand. However, it can overfit if not pruned properly.
3. **Random Forest** – An ensemble of multiple decision trees that improves accuracy and reduces overfitting. Works well with both small and large datasets.
4. **Support Vector Machine (SVM)** – Finds the best boundary to separate classes, even for complex data. The RBF kernel helps capture non-linear patterns effectively.

Performance Metrics Considered



```python
# Perform cross-validation (5-fold, or reduce if needed for small datasets)
cv_scores = cross_val_score(model, X_train, y_train, cv=5)

# Compute classification metrics
metrics = {
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred, average='weighted', zero_division=0),
    'recall': recall_score(y_test, y_pred, average='weighted', zero_division=0),
    'f1_score': f1_score(y_test, y_pred, average='weighted', zero_division=0),
    'cv_accuracy': np.mean(cv_scores),
}
```



```python
# Compute ROC-AUC for each class
roc_auc_scores = [roc_auc_score(y_test[:, i], y_pred_proba[:, i]) for i in range(y_test.shape[1])]
metrics['roc_auc'] = np.mean(roc_auc_scores)  # Average ROC-AUC over all classes

return metrics
```

Fig. 14. Performance metrics declaration

- **Accuracy**: Measures the overall proportion of correct predictions but can be misleading in imbalanced datasets.
- **Precision**: Focuses on the correctness of positive predictions, minimizing false positives.
- **Recall**: Measures the model's ability to identify actual positive cases, minimizing false negatives.
- **F1-Score**: Balances precision and recall, useful when there is an uneven class distribution.
- **Cross-Validation Accuracy**: Assesses model consistency across different data subsets to prevent overfitting.
- **ROC-AUC**: Evaluates the model's ability to distinguish between classes, with higher AUC indicating better class separation.

Here, we considered **F1 score** as the main evaluation criteria as it maintains the balance between both precision and recall for a model as it is the harmonic mean of precision and recall

Performance

Model	Accuracy	Precision	Recall	F1 Score	CV Accuracy	ROC AUC
Decision Tree	0.942478	0.943671	0.942478	0.94305	0.901247	0.849831
Random Forest	0.929204	0.931160	0.929204	0.93003	0.916388	0.90496
Support Vector Machine	0.564159	0.859689	0.564159	0.66533	0.652927	0.816173
Logistic Regression	0.407080	0.824193	0.407080	0.526678	0.534322	0.807080

Table. 3. Performance metrics

Data Modeling - Model Evaluation and Comparison

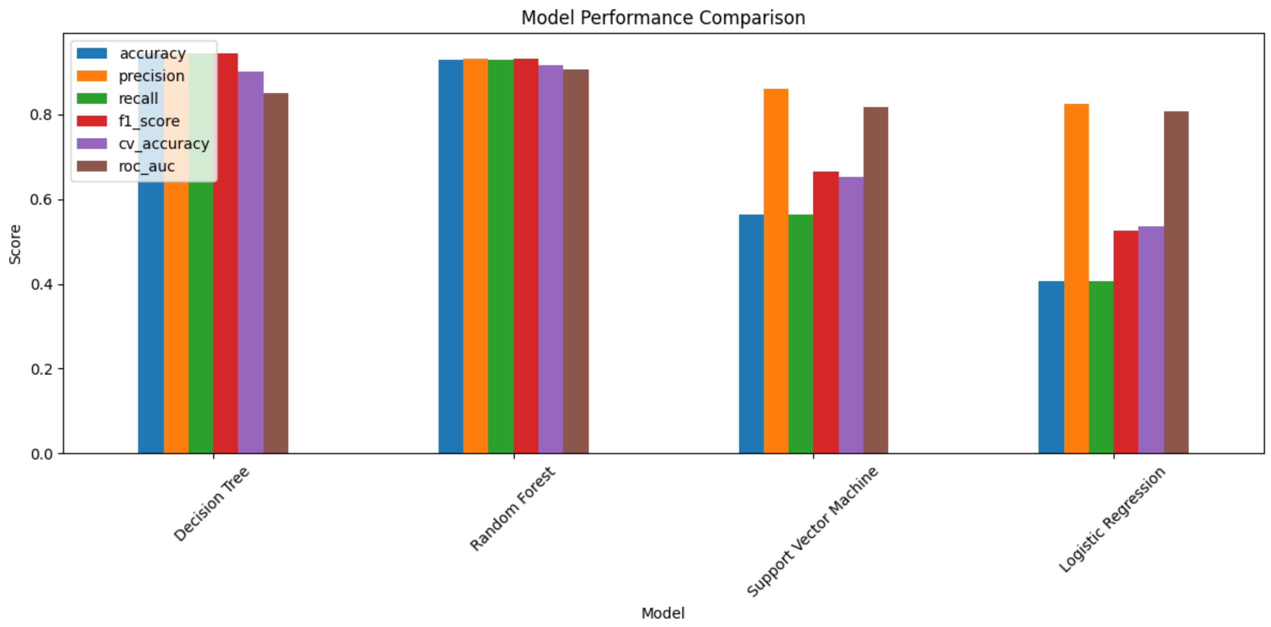
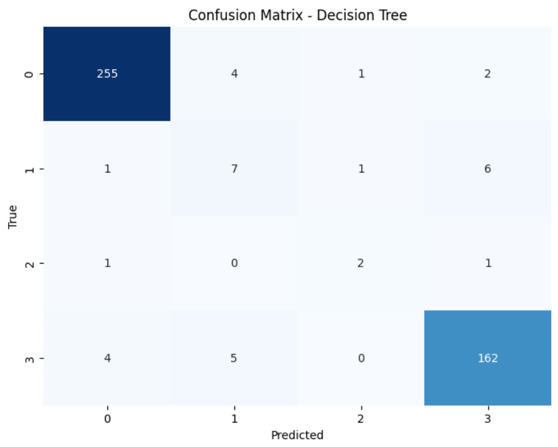


Fig. 15. Model comparison plot

Looking at the barplot, it is evident that Tree-based models Decision Tree and Random Forest performed well since all the six classification evaluation metrics are high.

Confusion Matrix Analysis (Decision Tree)



True \ Predicted	0	1	2	3
0	255	4	1	2
1	1	7	1	6
2	1	0	2	1
3	4	5	0	162

Fig. 16. Confusion matrix for Decision Tree

ROC Curve Analysis

Class 0 - AUC: 0.97, Class 1 - AUC: 0.72, Class 2 - AUC: 0.75, Class 3 - AUC: 0.96

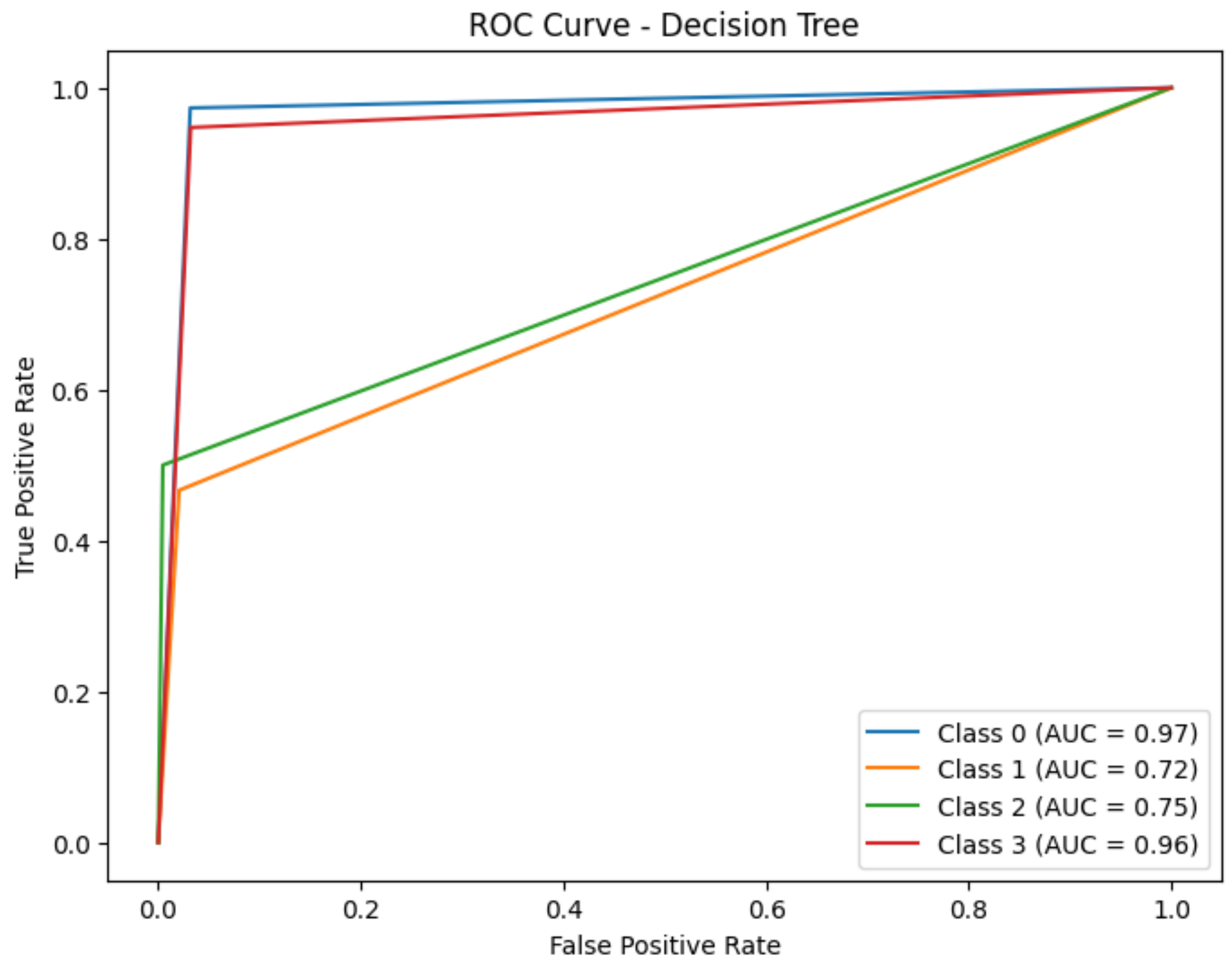


Fig. 17. ROC curve for Decision Tree

Decision Tree Model Selection

The Decision Tree model is chosen by considering all these factors:

- **F1-Score**: 94.3% – Balanced performance between precision and recall.
- **Accuracy**: 94.2% – Highest among the models.
- **Precision & Recall**: 94.4% & 94.2% – Very good at identifying the correct class without many false positives or misses.
- **Cross-Validation Accuracy**: 90.1% – Shows consistency across different data subsets, though slightly lower than training accuracy, hinting at some overfitting.
- **ROC AUC**: 85.0% – Strong ability to differentiate between classes, though Random Forest performs slightly better here.

Job Matching Results

The model implementation includes a job recommendation component using cosine similarity between course categories and job titles.

1. Job Recommendation based on the Course Title:
   - Data Science → Data Science Engineer, Data Science Manager, Data Analyst, Data Engineer
   - Business → Business Analyst, Business Consultant, Business Intelligence Analyst, Business System Analyst
   - Computer Science → Data Science Engineer, Data Science Manager, Software Engineer, Data Engineer
   - Information Technology → Software Engineer, Data Engineer, Business Analyst, Developer
2. Patterns:
   - "Software Engineer" and "Data Engineer" appear frequently as matches across diverse categories
   - Technical job titles dominate the recommendations across most categories

Conclusion

The Decision Tree model stands out as the best-performing model in this analysis, with an impressive accuracy of 94.2%, and excellent precision (94.4%) and recall (94.2%), ensuring it identifies the correct class with minimal false positives or misses. The balanced F1-score of 94.3% further reinforces its robust performance. Although there is some slight overfitting, as indicated by the lower cross-validation accuracy of 90.1%, the model demonstrates good generalization, with a strong ROC AUC of 85.0%.

In terms of job matching, the cosine similarity approach effectively aligns course categories with relevant job titles. Notably, roles like "Software Engineer" and "Data Engineer" appear frequently across various categories, reflecting the high demand for technical expertise. The results emphasize the importance of technical job titles in most fields, offering valuable guidance for students seeking career opportunities based on their educational background.