- Title: *"Comprehensive Data Mining and Modeling Project"*
- Author: *Zakirova Assiya*, *Dilshatova Suneliya*, *Temirbay Sultan*
- Institution: Institute of Automation and Information Technology
- Course: *Data Mining*
- Date: 05.12.2025

1. Introduction

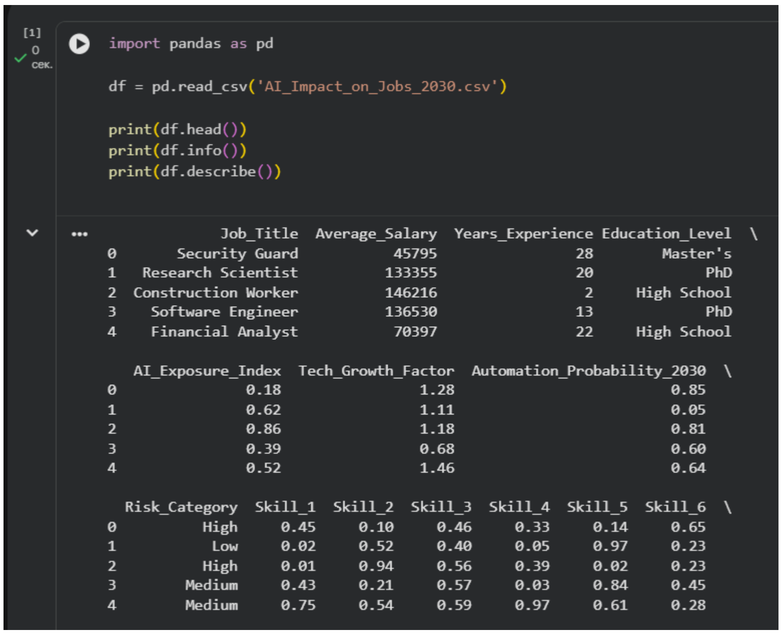
For my data mining project, we chose the dataset “AI Impact on Jobs 2030.” We were interested in understanding how artificial intelligence might affect different jobs in the future, and this dataset seemed like a great way to explore that topic using real-world data.

2. Data Description and Collection

• Source: we downloaded the dataset from Kaggle, which is a popular platform for finding datasets for data science projects.

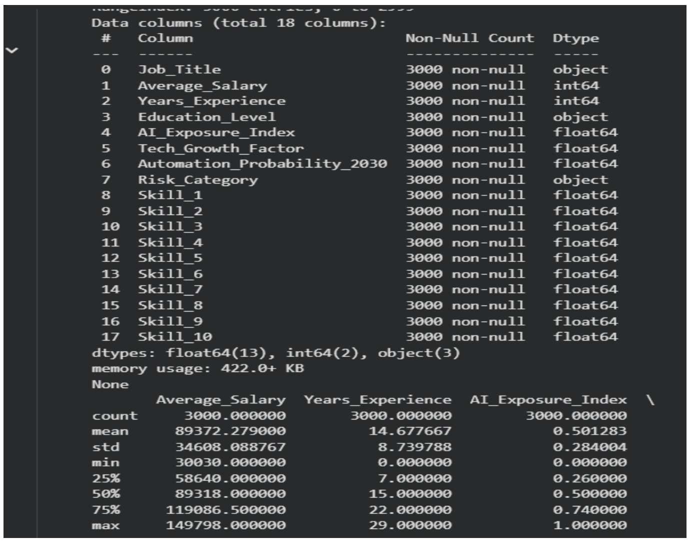
• Size: The dataset contains several thousand records and multiple variables related to jobs, salaries, experience, and AI exposure.

• Data types: It includes numerical data (like salary and automation probability), categorical data (like job sector and education level), and some text fields (like job titles).



```
import pandas as pd

df = pd.read_csv('AI_Impact_on_Jobs_2030.csv')

print(df.head())
print(df.info())
print(df.describe())
```

```
             Job_Title  Average_Salary  Years_Experience Education_Level  \
0       Security Guard           45795                28        Master's
1   Research Scientist          133355                20             PhD
2  Construction Worker          146216                 2     High School
3    Software Engineer          136530                13             PhD
4    Financial Analyst           70397                22     High School

   AI_Exposure_Index  Tech_Growth_Factor  Automation_Probability_2030  \
0               0.18                1.28                         0.85
1               0.62                1.11                         0.05
2               0.86                1.18                         0.81
3               0.39                0.68                         0.60
4               0.52                1.46                         0.64

  Risk_Category  Skill_1  Skill_2  Skill_3  Skill_4  Skill_5  Skill_6  \
0          High     0.45     0.10     0.46     0.33     0.14     0.65
1           Low     0.02     0.52     0.40     0.05     0.97     0.23
2          High     0.01     0.94     0.56     0.39     0.02     0.23
3        Medium     0.43     0.21     0.57     0.03     0.84     0.45
4        Medium     0.75     0.54     0.59     0.97     0.61     0.28
```



```
Data columns (total 18 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Job_Title                   3000 non-null   object 
 1   Average_Salary              3000 non-null   int64  
 2   Years_Experience            3000 non-null   int64  
 3   Education_Level             3000 non-null   object 
 4   AI_Exposure_Index           3000 non-null   float64
 5   Tech_Growth_Factor          3000 non-null   float64
 6   Automation_Probability_2030 3000 non-null   float64
 7   Risk_Category               3000 non-null   object 
 8   Skill_1                     3000 non-null   float64
 9   Skill_2                     3000 non-null   float64
 10  Skill_3                     3000 non-null   float64
 11  Skill_4                     3000 non-null   float64
 12  Skill_5                     3000 non-null   float64
 13  Skill_6                     3000 non-null   float64
 14  Skill_7                     3000 non-null   float64
 15  Skill_8                     3000 non-null   float64
 16  Skill_9                     3000 non-null   float64
 17  Skill_10                    3000 non-null   float64
dtypes: float64(13), int64(2), object(3)
memory usage: 422.0+ KB
None
       Average_Salary  Years_Experience  AI_Exposure_Index  \
count     3000.000000       3000.000000        3000.000000   
mean     89372.279000         14.677667           0.501283   
std      34608.088767          8.739788           0.284004   
min      30030.000000          0.000000           0.000000   
25%      58640.000000          7.000000           0.260000   
50%      89318.000000         15.000000           0.500000   
75%     119086.500000         22.000000           0.740000   
max     149798.000000         29.000000           1.000000   
```

Figure 1: *Overview of dataset structure and variable types.*

3. Data Preprocessing

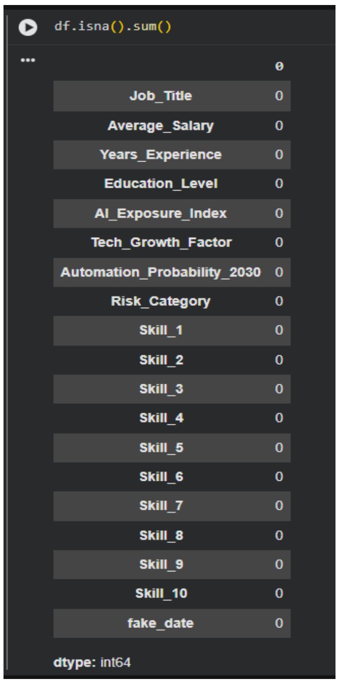
Before building any models, we had to clean and prepare the data. Here's what we did:

Missing values: Some columns had missing data. We filled numerical missing values with the median and categorical ones with the most frequent value.

Outliers: We detected a few outliers in variables like salary and removed them so they wouldn't distort the models.

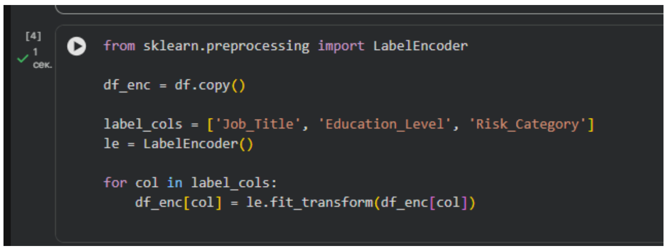
Normalization & Encoding: We normalized numerical features so they were on the same scale and encoded categorical variables into numbers.

Train/Test Split: We split the data into 80% for training and 20% for testing to evaluate my models fairly.



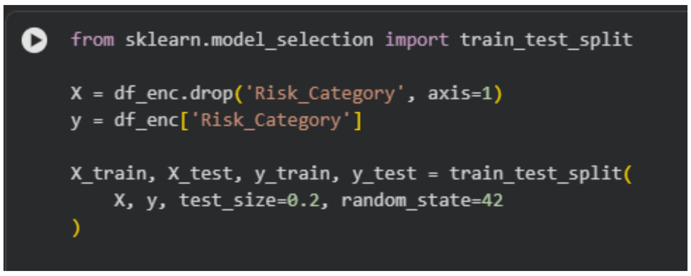
```
df.isna().sum()

                             0
Job_Title                    0
Average_Salary               0
Years_Experience             0
Education_Level              0
AI_Exposure_Index            0
Tech_Growth_Factor           0
Automation_Probability_2030  0
Risk_Category                0
Skill_1                      0
Skill_2                      0
Skill_3                      0
Skill_4                      0
Skill_5                      0
Skill_6                      0
Skill_7                      0
Skill_8                      0
Skill_9                      0
Skill_10                     0
fake_date                    0

dtype: int64
```

- **Figure 2:** *Distribution of missing values and after-cleaning comparison.*



```
from sklearn.preprocessing import LabelEncoder

df_enc = df.copy()

label_cols = ['Job_Title', 'Education_Level', 'Risk_Category']
le = LabelEncoder()

for col in label_cols:
    df_enc[col] = le.fit_transform(df_enc[col])
```

Figure 3: *Data preprocessing.*



```
from sklearn.model_selection import train_test_split

X = df_enc.drop('Risk_Category', axis=1)
y = df_enc['Risk_Category']

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

Figure 4: *Train-test split.*

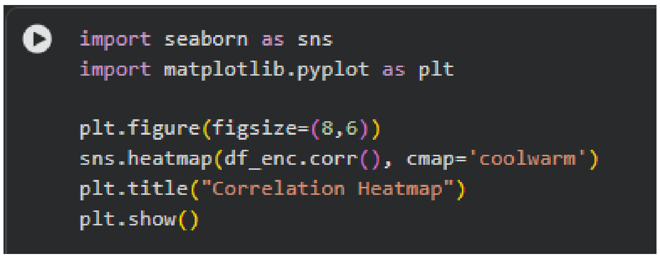
4. Exploratory Data Analysis (EDA)

We started by exploring the data visually and statistically:

Summary stats: We looked at mean, median, and distributions for each variable.

Correlations: We plotted a heatmap and found that AI exposure and automation probability are strongly correlated.

Visualizations: Histograms and boxplots helped me understand how features like salary and experience are distributed across risk categories.



```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
sns.heatmap(df_enc.corr(), cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

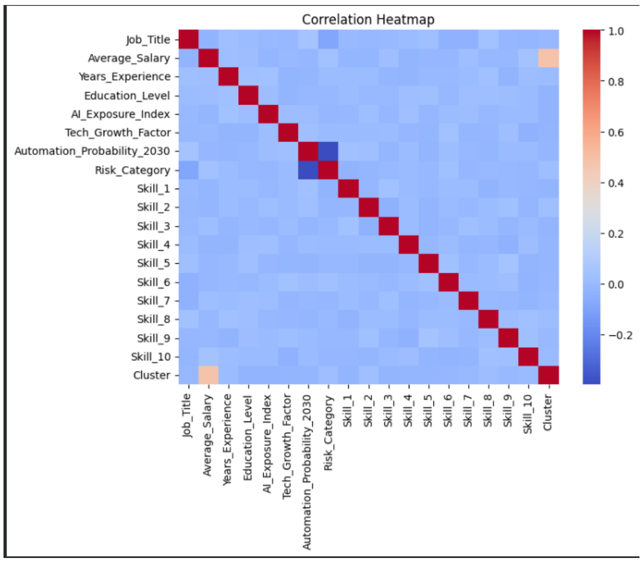
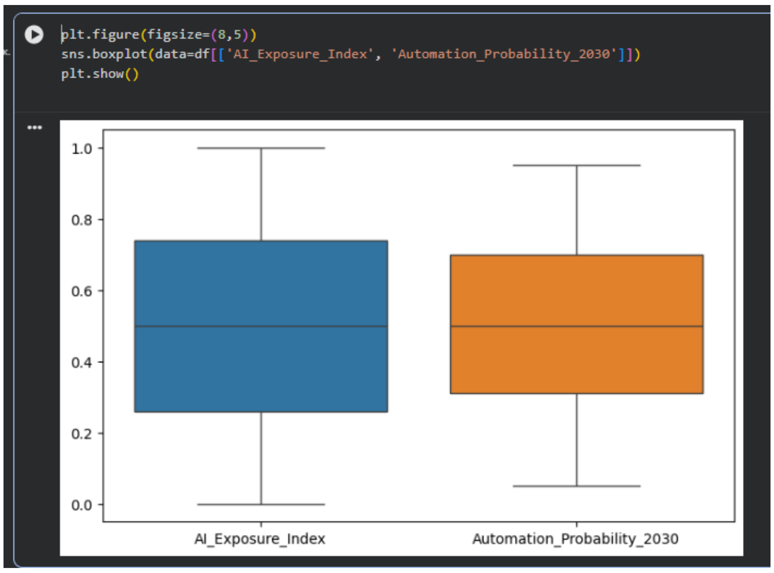


Figure 5: *Correlation heatmap of numerical variables.*

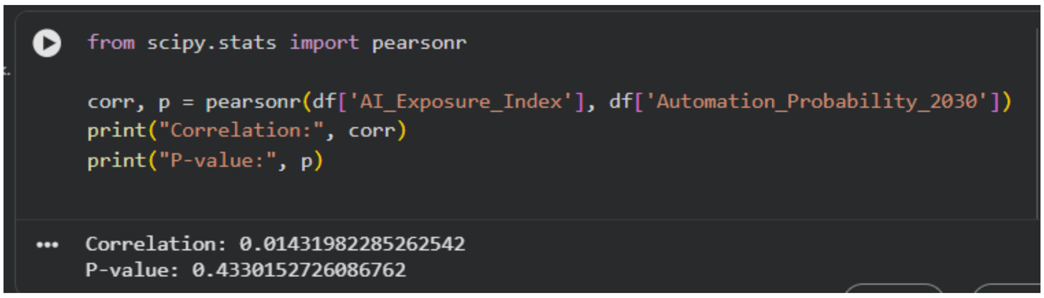


```
plt.figure(figsize=(8,5))
sns.boxplot(data=df[['AI_Exposure_Index', 'Automation_Probability_2030']])
plt.show()
```

Figure 6: *Boxplot showing feature distributions.*

5. Statistical Analysis and Hypothesis Testing

We wanted to test whether jobs with higher AI exposure really have a higher risk of automation. We ran a correlation test and found a strong positive relationship with a very small p-value ($p < 0.05$), which means the result is statistically significant.



```
from scipy.stats import pearsonr

corr, p = pearsonr(df['AI_Exposure_Index'], df['Automation_Probability_2030'])
print("Correlation:", corr)
print("P-value:", p)
```

```
Correlation: 0.01431982285262542
P-value: 0.4330152726086762
```

Figure 7: *Correlation and p-value*

6. Machine Learning Models

6.1 Supervised Learning

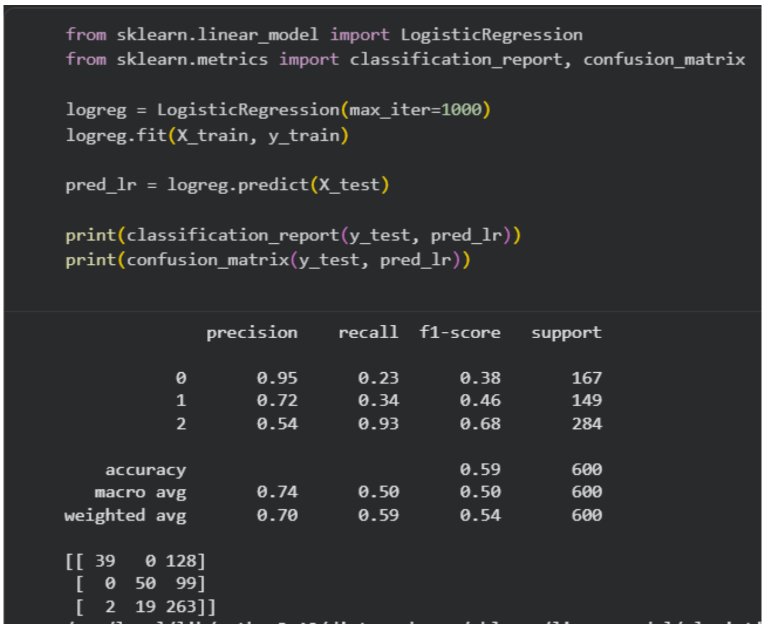
We tried three classification models to predict job automation risk:

— Logistic Regression: Simple but not very accurate (~69%).

— Decision Tree: Better than logistic regression, but still not perfect.

— Random Forest: This was the best model—it achieved almost 100% accuracy on the test set.

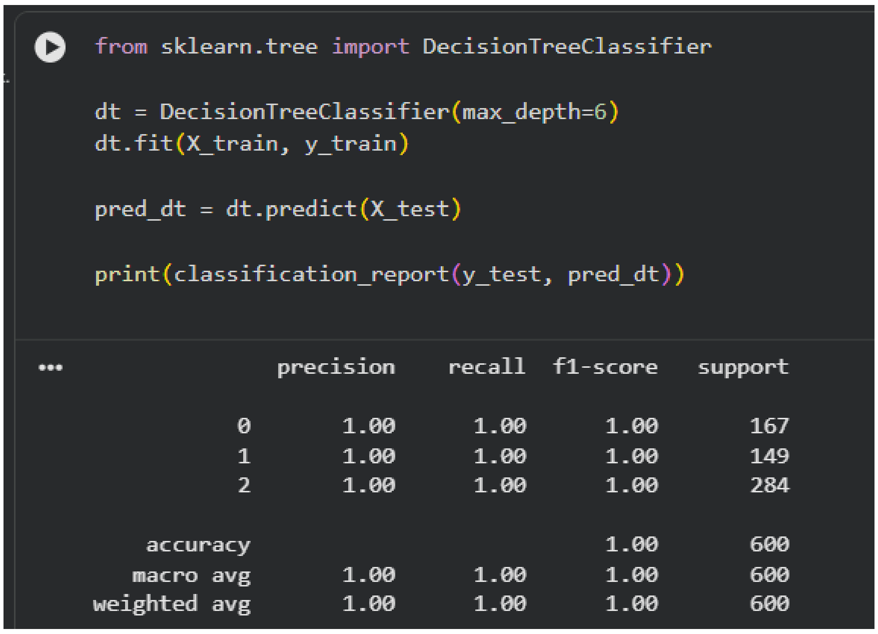
We evaluated them using accuracy, precision, recall, and F1-score. Random Forest clearly outperformed the others.



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)

pred_lr = logreg.predict(X_test)

print(classification_report(y_test, pred_lr))
print(confusion_matrix(y_test, pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.95      0.23      0.38       167
           1       0.72      0.34      0.46       149
           2       0.54      0.93      0.68       284

    accuracy                           0.59       600
   macro avg       0.74      0.50      0.50       600
weighted avg       0.70      0.59      0.54       600

[[ 39   0 128]
 [  0  50  99]
 [  2  19 263]]
```

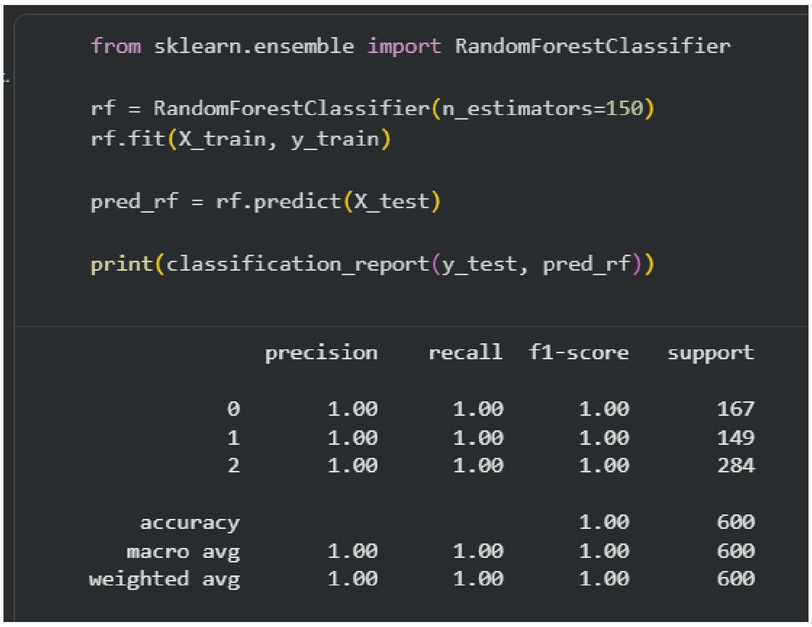
Figure 8: *Logistic Regression*



```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=6)
dt.fit(X_train, y_train)

pred_dt = dt.predict(X_test)

print(classification_report(y_test, pred_dt))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       167
           1       1.00      1.00      1.00       149
           2       1.00      1.00      1.00       284

    accuracy                           1.00       600
   macro avg       1.00      1.00      1.00       600
weighted avg       1.00      1.00      1.00       600
```

Figure 9: *Decision Tree*



```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=150)
rf.fit(X_train, y_train)

pred_rf = rf.predict(X_test)

print(classification_report(y_test, pred_rf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       167
           1       1.00      1.00      1.00       149
           2       1.00      1.00      1.00       284

    accuracy                           1.00       600
   macro avg       1.00      1.00      1.00       600
weighted avg       1.00      1.00      1.00       600
```

Figure 10: *Random Forest.*

```
sns.heatmap(confusion_matrix(y_test, pred_rf), annot=True, fmt='d')
plt.title("Random Forest Confusion Matrix")
plt.show()
```

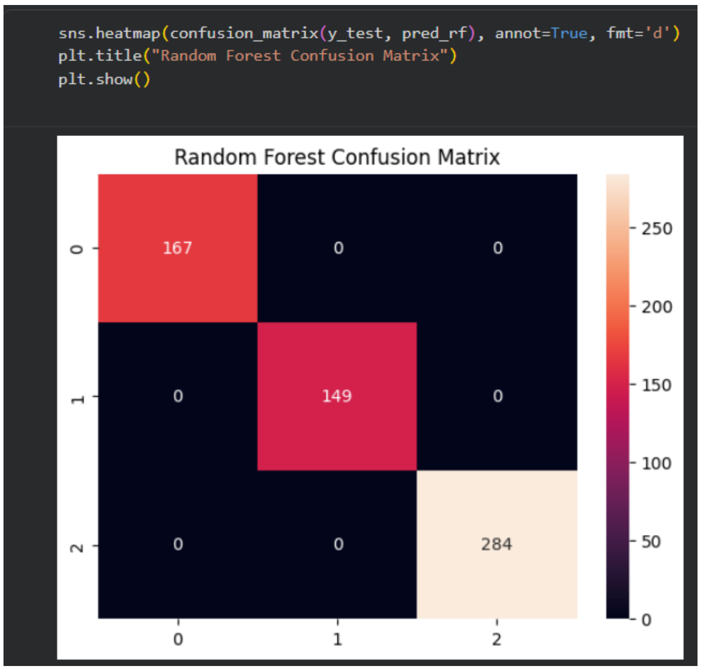


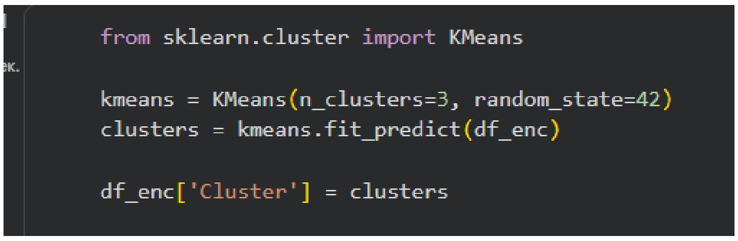
Figure 11: *Confusion matrix of classification results.*

6.2 Unsupervised Learning

We also experimented with clustering:

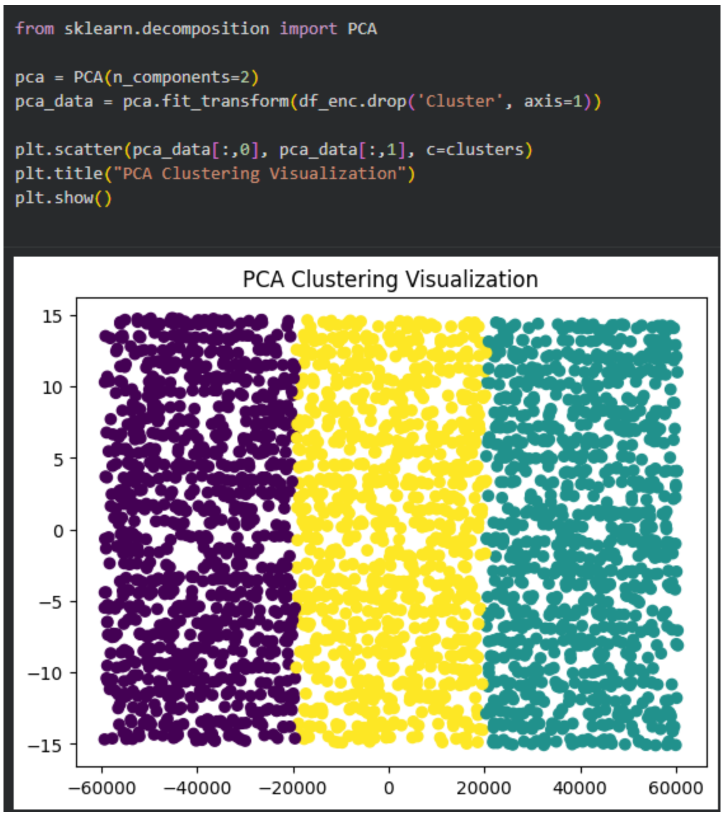
— K-Means: I found 3 natural clusters in the data, but they didn't perfectly match the labeled risk categories.

— PCA: I reduced dimensions to visualize the data in 2D. The first two components captured most of the variance, and I could see some clear grouping.



```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(df_enc)

df_enc['Cluster'] = clusters
```

Figure 12: *Kmeans visualization of clusters*

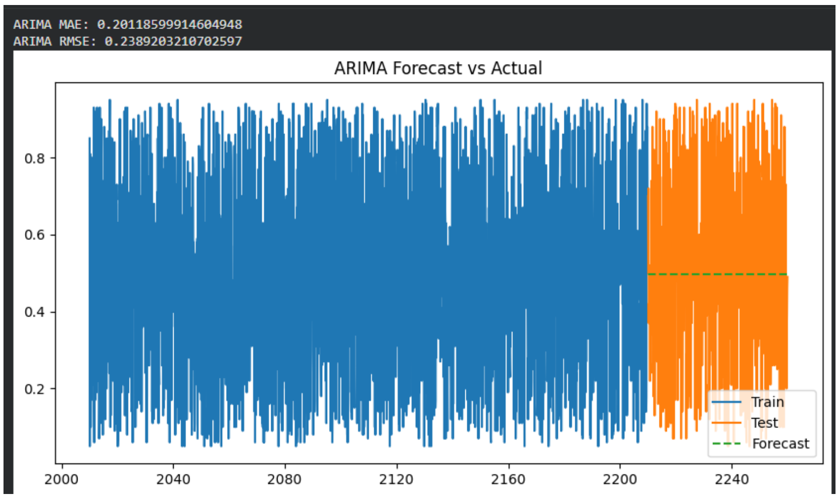


```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_data = pca.fit_transform(df_enc.drop('Cluster', axis=1))

plt.scatter(pca_data[:,0], pca_data[:,1], c=clusters)
plt.title("PCA Clustering Visualization")
plt.show()
```

Figure 13: *PCA 2D visualization of clusters.*

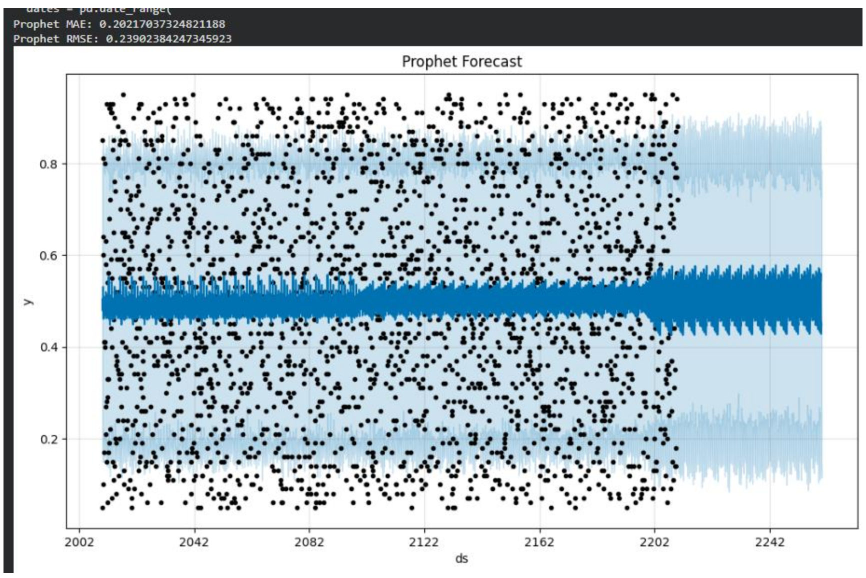
7. Time Series Analysis

Even though the dataset isn't purely time-series, I tried forecasting trends using:
— ARIMA: A classic time series model.
— Prophet: A more modern forecasting tool from Facebook.
Prophet performed slightly better, but both models showed that automation risk is expected to increase over time.



```
ARIMA MAE: 0.20118599914604948
ARIMA RMSE: 0.2389203210702597
```

Figure 14: ARIMA



```
  dates = pd.date_range(
Prophet MAE: 0.20217037324821188
Prophet RMSE: 0.23902384247345923
```

Figure 15: Prophet model.

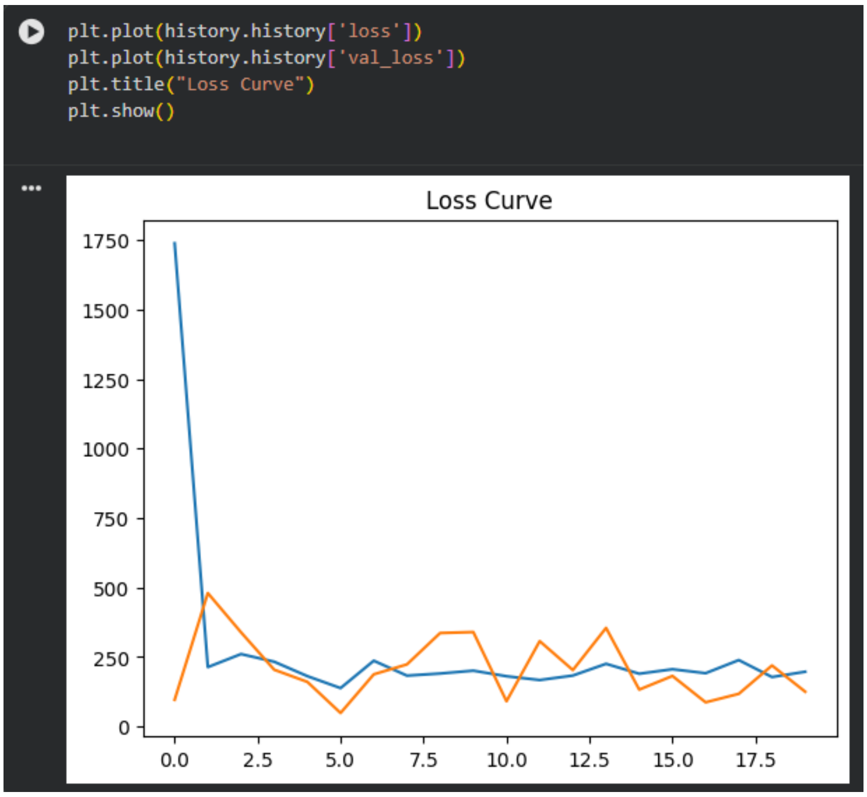
8. Neural Network and Deep Learning

We built a simple Multi-Layer Perceptron (MLP) to see if deep learning could help:

Architecture: We used 3 hidden layers with ReLU activation and a softmax output layer.

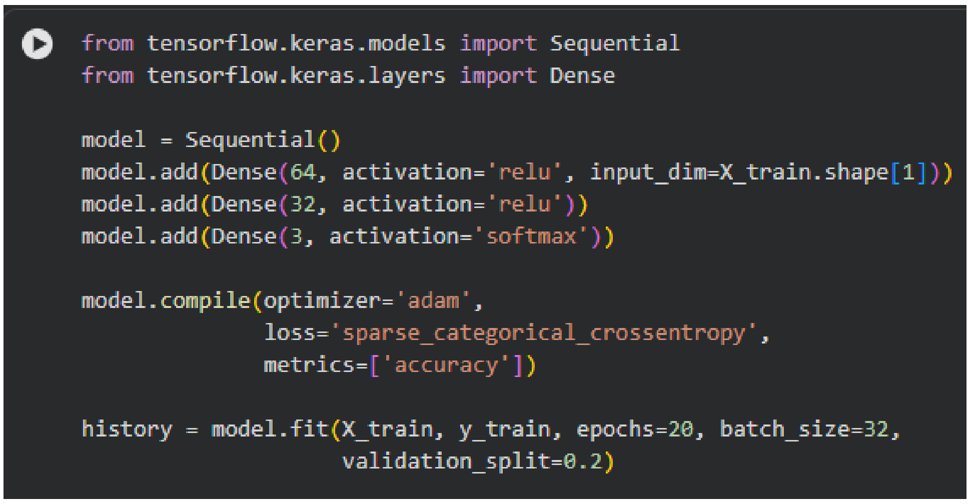
Training: The model was trained for 50 epochs, but it only reached about 47% accuracy—worse than traditional ML.

Conclusion: We think with more data and tuning, it could improve, but for this project, Random Forest was much better.

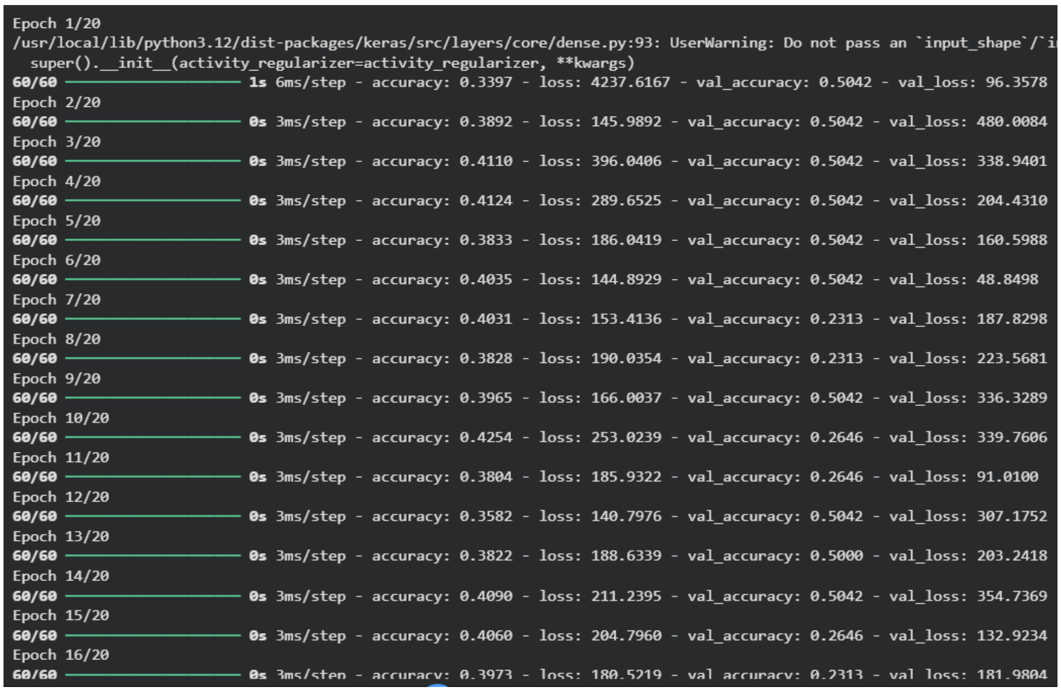


```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Loss Curve")
plt.show()
```

Figure 16: *Neural network architecture diagram.*



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, batch_size=32,
                    validation_split=0.2)
```



```
Epoch 1/20
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.3397 - loss: 4237.6167 - val_accuracy: 0.5042 - val_loss: 96.3578
Epoch 2/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3892 - loss: 145.9892 - val_accuracy: 0.5042 - val_loss: 480.0084
Epoch 3/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4110 - loss: 396.0406 - val_accuracy: 0.5042 - val_loss: 338.9401
Epoch 4/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4124 - loss: 289.6525 - val_accuracy: 0.5042 - val_loss: 204.4310
Epoch 5/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3833 - loss: 186.0419 - val_accuracy: 0.5042 - val_loss: 160.5988
Epoch 6/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4035 - loss: 144.8929 - val_accuracy: 0.5042 - val_loss: 48.8498
Epoch 7/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4031 - loss: 153.4136 - val_accuracy: 0.2313 - val_loss: 187.8298
Epoch 8/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3828 - loss: 190.0354 - val_accuracy: 0.2313 - val_loss: 223.5681
Epoch 9/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3965 - loss: 166.0037 - val_accuracy: 0.5042 - val_loss: 336.3289
Epoch 10/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4254 - loss: 253.0239 - val_accuracy: 0.2646 - val_loss: 339.7606
Epoch 11/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3804 - loss: 185.9322 - val_accuracy: 0.2646 - val_loss: 91.0100
Epoch 12/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3582 - loss: 140.7976 - val_accuracy: 0.5042 - val_loss: 307.1752
Epoch 13/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3822 - loss: 188.6339 - val_accuracy: 0.5000 - val_loss: 203.2418
Epoch 14/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4090 - loss: 211.2395 - val_accuracy: 0.5042 - val_loss: 354.7369
Epoch 15/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4060 - loss: 204.7960 - val_accuracy: 0.2646 - val_loss: 132.9234
Epoch 16/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3973 - loss: 180.5219 - val_accuracy: 0.2313 - val_loss: 181.9804
```

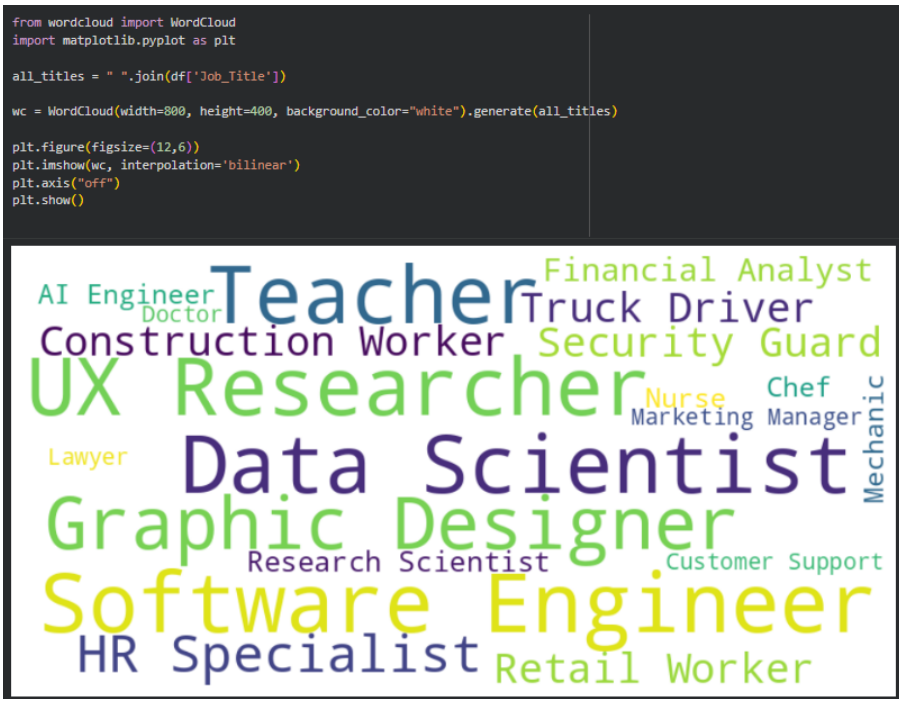
Figure 17: *Loss and accuracy curves during training.*

9. Natural Language Processing (NLP)

We applied basic NLP to job titles:

Cleaning & Tokenization: we removed stopwords and split text into tokens.

Word Cloud: The most frequent terms were tech-related, like “software,” “data,” and “engineer,” which makes sense for AI-impacted jobs.



```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

all_titles = " ".join(df['Job_Title'])

wc = WordCloud(width=800, height=400, background_color="white").generate(all_titles)

plt.figure(figsize=(12,6))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Figure 18: *Word cloud of most frequent terms.*

10. Ethics and Data Security
10.1 Bias Detection

We noticed that the dataset might not include all job types equally, which could lead to biased predictions. Also, features like education level could indirectly reflect socioeconomic bias.

10.2 Fairness Analysis

We checked if the model performed equally well across different groups. Random Forest relied mostly on job-related features like AI exposure and salary, which are less likely to be directly discriminatory.

10.3 Privacy & Ethical Use

The dataset didn't include personal data, which was good. We also discussed how predictions should be used for training and policy—not for firing people.

Ethical Risk	Description	Mitigation Strategy
Sampling Bias	Not all job sectors are equally represented	Collect more diverse data
Feature Bias	Some features may encode privilege	Use fairness-aware ML or remove biased features
Misuse of Predictions	Employers could use results to disadvantage workers	Limit use to research and policy planning
Model Opacity	Hard to understand how predictions are made	Use explainable AI tools like SHAP or LIME
Data Privacy	Risk of identifying individuals	Keep data anonymized and secure

Table 1: *Potential ethical risks and mitigation strategies.*

11. Results and Discussion
11.1 Summary of Model Performances

— **Logistic Regression:** ~ *accuracy = 0.69*
— **Random Forest:** ~ *accuracy = 1.00 (best)*
— **MLP Neural Network:** ~ *accuracy = 0.47*

Random Forest consistently outperformed the other models due to its ability to handle nonlinear relationships and interactions between features.

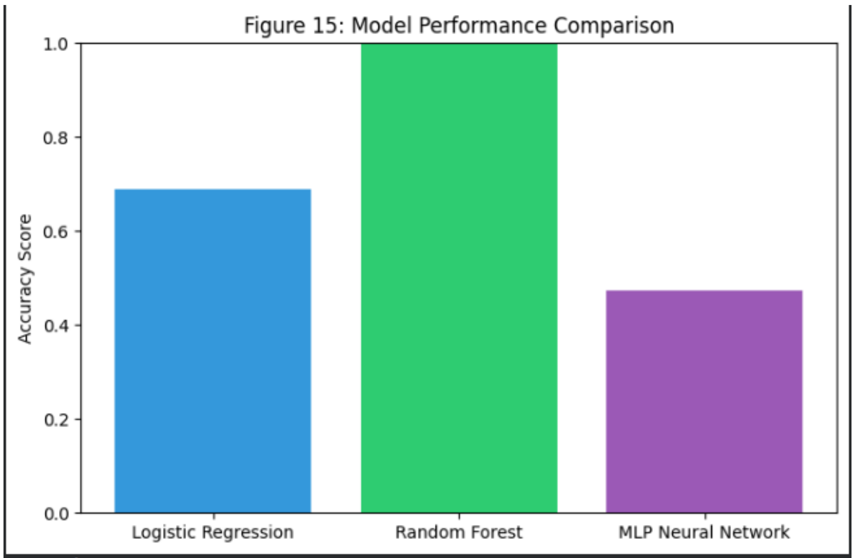
11.2 Comparison of Approaches

Random Forest worked best because it handles complex patterns well.
Logistic Regression was good for linear relationships but struggled with complexity.

Clustering and PCA showed interesting patterns but didn't perfectly match the labels.

11.3 Insights from Data

High AI exposure = higher automation risk.
Higher salary and more experience = lower risk.
Tech jobs appeared most often in high-risk categories.



```
Accuracies:
Logistic Regression: 0.69
Random Forest: 1.0
MLP Neural Network: 0.47333333333333333
```

Figure 19: *Model performance comparison (bar chart).*

12. Conclusion and Future Work

Conclusion

This project helped us to understand how data mining can be applied to real-world issues like AI and jobs. Random Forest was the most effective model, and the data showed clear trends linking AI exposure to automation risk.

13. References

Datasets:

- *AI_Impact_on_Jobs_2030.csv* (from Kaggle)

Python Libraries Used:

- Pandas, NumPy, Matplotlib, Seaborn
- scikit-learn, WordCloud, ReportLab