Import Required Libraries

In [1]:
```
# Data handling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Preprocessing & splitting
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.impute import SimpleImputer
from sklearn.utils import resample

# Evaluation metrics
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_scor
    classification_report,confusion_matrix,auc,roc_auc_score, roc_curve
)

# Regression Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier

# XGBoost
from xgboost import XGBClassifier

# Warnings
import warnings
warnings.filterwarnings("ignore")

# Save the model
import joblib
```

Load Dataset

In [2]:
```
data = pd.read_csv('AI_Impact_on_Jobs_2030.csv')
data.head()
```

Out[2]:

	Job_Title	Average_Salary	Years_Experience	Education_Level	AI_Exposure_Index
0	Security Guard	45795	28	Master's	0.18
1	Research Scientist	133355	20	PhD	0.62
2	Construction Worker	146216	2	High School	0.86
3	Software Engineer	136530	13	PhD	0.39
4	Financial Analyst	70397	22	High School	0.52

Data Preprocessing

In [3]:
```
data.duplicated().sum()
```

Out[3]: 0

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
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
```

In [5]:
```
data.drop(['Skill_1', 'Skill_2',
       'Skill_3', 'Skill_4', 'Skill_5', 'Skill_6', 'Skill_7', 'Skill_8',
       'Skill_9', 'Skill_10'],axis=1,inplace=True)
```

In [6]:
```
data.describe()
```

Out[6]:

	Average_Salary	Years_Experience	AI_Exposure_Index	Tech_Growth_Factor	Auton
count	3000.000000	3000.000000	3000.000000	3000.000000	
mean	89372.279000	14.677667	0.501283	0.995343	
std	34608.088767	8.739788	0.284004	0.287669	
min	30030.000000	0.000000	0.000000	0.500000	
25%	58640.000000	7.000000	0.260000	0.740000	
50%	89318.000000	15.000000	0.500000	1.000000	
75%	119086.500000	22.000000	0.740000	1.240000	
max	149798.000000	29.000000	1.000000	1.500000	

Encoding

In [7]:
```
data['Education_Level'] = data['Education_Level'].map({"Master's": 2, 'PhD': 3,
```

In [8]:
```
data['Risk_Category']= data['Risk_Category'].map({'High':2, 'Low':0, 'Medium':1}
```

In [9]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Job_Title'] = le.fit_transform(data['Job_Title'])
```

In [10]:
```
data['Risk_Category'].value_counts()
```

Out[10]:
```
Risk_Category
1    1521
2     740
0     739
Name: count, dtype: int64
```

Splitting for Target and Feature into Train Test Split

In [11]:
```
X = data.drop(['Risk_Category'],axis=1)
y = data['Risk_Category']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.20,random_state=

scaler = StandardScaler()

#scale features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled.shape,X_test_scaled.shape
```

Out[11]:
```
((2400, 7), (600, 7))
```

Preprocessing pipeline

In [12]:
```
preprocessor = Pipeline([
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])
```

Define models & parameter grids for classification

In [13]:
```
models = {
    # -------------------------------
    "Logistic Regression": {
        "model": LogisticRegression(solver='lbfgs', max_iter=500, random_state=4
        "param_grid": {
            "C": [0.1, 1, 10],
            "penalty": ['l2']
        }
    },
    # -------------------------------
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "param_grid": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4]
        }
    },
    # -------------------------------
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # -------------------------------
    "Gradient Boosting": {
        "model": GradientBoostingClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "learning_rate": [0.01, 0.1, 0.2],
            "max_depth": [3, 5]
        }
    },
    # -------------------------------
    "AdaBoost": {
        "model": AdaBoostClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [50, 100, 200],
            "learning_rate": [0.01, 0.1, 1.0]
        }
```

```
    },
    # ------------------------------
    "Extra Trees": {
        "model": ExtraTreesClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "K-Nearest Neighbors": {
        "model": KNeighborsClassifier(),
        "param_grid": {
            "n_neighbors": [3, 5, 7, 9],
            "weights": ["uniform", "distance"],
            "p": [1, 2]  # Manhattan (L1) or Euclidean (L2)
        }
    },
    # ------------------------------
    "MLP Classifier": {
        "model": MLPClassifier(random_state=42),
        "param_grid": {
            "hidden_layer_sizes": [(50,), (100,)],
            "activation": ["relu", "tanh"],
            "solver": ["adam"],
            "max_iter": [200]
        }
    },
    # ------------------------------
    "XGBoost": {
        "model": XGBClassifier(use_label_encoder=False, eval_metric="logloss", r
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [3, 5, 7],
            "learning_rate": [0.01, 0.1, 0.2],
            "subsample": [0.8, 1.0],
            "colsample_bytree": [0.8, 1.0]
        }
    }
}

print(f" Total Models Loaded: {len(models)}")
```

```
Total Models Loaded: 9
```

Model Evaluation

In [14]:

```
results = []
best_score = -float('inf')
best_model = None
best_name = None

# Run GridSearchCV for each classification model
for name, mp in models.items():
    print(f"\n Running GridSearchCV for {name}...")
    grid = GridSearchCV(mp['model'], param_grid=mp['param_grid'], cv=5, scoring=
```

```
    grid.fit(X_train_scaled, y_train)

    y_pred = grid.predict(X_test_scaled)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        'Model': name,
        'Best Params': grid.best_params_,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1 Score': f1
    })

    print(f"{name} Best CV Score (Accuracy): {grid.best_score_:.4f}")
    print(f"{name} Best Params: {grid.best_params_}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred


    if grid.best_score_ > best_score:
        best_score = grid.best_score_
        best_model = grid.best_estimator_
        best_name = name

# Output summary
results_df = pd.DataFrame(results)
print("\n Summary Results:")
print(results_df)

print(f"\n Best Model: {best_name}")
print(f" Best CV Score (Accuracy): {best_score:.4f}")
```

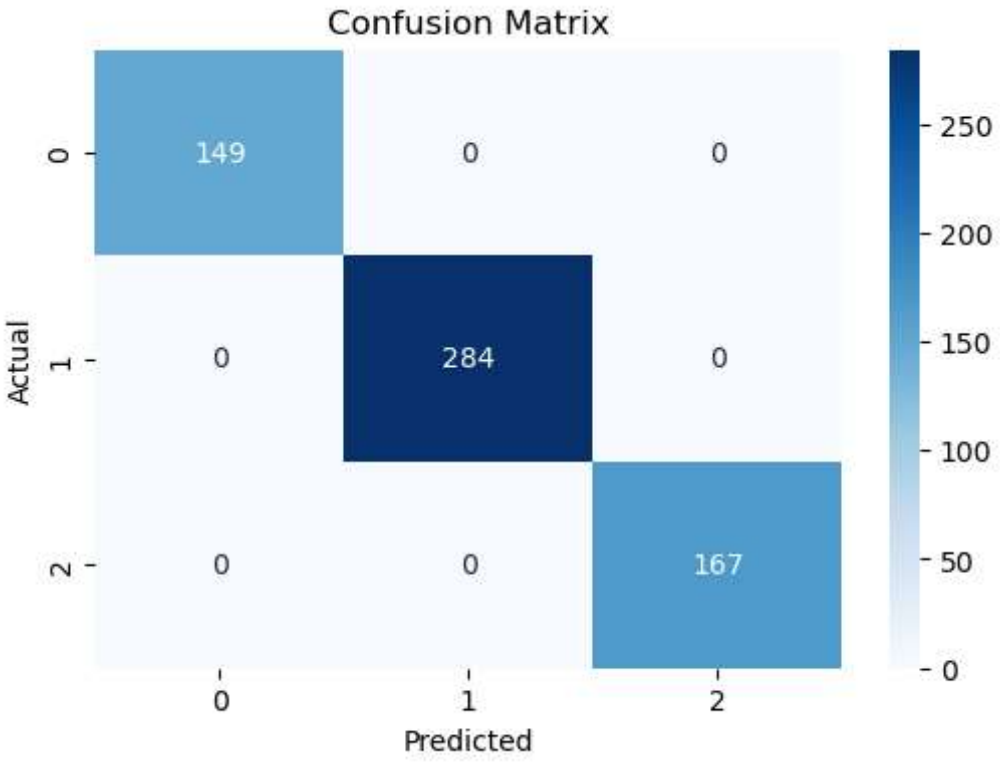
```
 Running GridSearchCV for Logistic Regression...
Logistic Regression Best CV Score (Accuracy): 0.9950
Logistic Regression Best Params: {'C': 10, 'penalty': 'l2'}
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       149
           1       0.99      1.00      0.99       284
           2       1.00      0.99      0.99       167

    accuracy                           0.99       600
   macro avg       1.00      0.99      1.00       600
weighted avg       1.00      0.99      1.00       600


 Running GridSearchCV for Decision Tree...
Decision Tree Best CV Score (Accuracy): 1.0000
Decision Tree Best Params: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_s
plit': 2}
Decision Tree Classification Report:
              precision    recall  f1-score   support

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

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


 Running GridSearchCV for Random Forest...
Random Forest Best CV Score (Accuracy): 1.0000
Random Forest Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2, 'n_estimators': 100}
Random Forest Classification Report:
              precision    recall  f1-score   support

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

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


 Running GridSearchCV for Gradient Boosting...
Gradient Boosting Best CV Score (Accuracy): 1.0000
Gradient Boosting Best Params: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimat
ors': 100}
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

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

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

```
weighted avg       1.00      1.00      1.00       600


 Running GridSearchCV for AdaBoost...
AdaBoost Best CV Score (Accuracy): 1.0000
AdaBoost Best Params: {'learning_rate': 0.01, 'n_estimators': 50}
AdaBoost Classification Report:
              precision    recall  f1-score   support

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

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


 Running GridSearchCV for Extra Trees...
Extra Trees Best CV Score (Accuracy): 0.9946
Extra Trees Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_
split': 2, 'n_estimators': 200}
Extra Trees Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       149
           1       0.99      0.99      0.99       284
           2       0.99      0.99      0.99       167

    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600


 Running GridSearchCV for K-Nearest Neighbors...
K-Nearest Neighbors Best CV Score (Accuracy): 0.9517
K-Nearest Neighbors Best Params: {'n_neighbors': 9, 'p': 1, 'weights': 'distanc
e'}
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       149
           1       0.94      0.95      0.95       284
           2       0.95      0.95      0.95       167

    accuracy                           0.95       600
   macro avg       0.95      0.95      0.95       600
weighted avg       0.95      0.95      0.95       600


 Running GridSearchCV for MLP Classifier...
MLP Classifier Best CV Score (Accuracy): 0.9925
MLP Classifier Best Params: {'activation': 'tanh', 'hidden_layer_sizes': (100,),
'max_iter': 200, 'solver': 'adam'}
MLP Classifier Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       149
           1       0.99      1.00      0.99       284
           2       0.99      0.98      0.99       167
```

```
    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600


 Running GridSearchCV for XGBoost...
XGBoost Best CV Score (Accuracy): 1.0000
XGBoost Best Params: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_dept
h': 3, 'n_estimators': 100, 'subsample': 0.8}
XGBoost Classification Report:
              precision    recall  f1-score   support

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

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


 Summary Results:
                Model                                        Best Params  \
0  Logistic Regression                          {'C': 10, 'penalty': 'l2'}
1        Decision Tree  {'max_depth': 3, 'min_samples_leaf': 1, 'min_s...
2        Random Forest  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
3    Gradient Boosting  {'learning_rate': 0.01, 'max_depth': 3, 'n_est...
4             AdaBoost        {'learning_rate': 0.01, 'n_estimators': 50}
5          Extra Trees  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
6  K-Nearest Neighbors  {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
7       MLP Classifier  {'activation': 'tanh', 'hidden_layer_sizes': (...
8              XGBoost  {'colsample_bytree': 0.8, 'learning_rate': 0.0...

   Accuracy  Precision    Recall  F1 Score
0  0.995000   0.995052  0.995000  0.995000
1  1.000000   1.000000  1.000000  1.000000
2  1.000000   1.000000  1.000000  1.000000
3  1.000000   1.000000  1.000000  1.000000
4  1.000000   1.000000  1.000000  1.000000
5  0.990000   0.990066  0.990000  0.990000
6  0.948333   0.948359  0.948333  0.948342
7  0.991667   0.991716  0.991667  0.991666
8  1.000000   1.000000  1.000000  1.000000

 Best Model: Decision Tree
 Best CV Score (Accuracy): 1.0000
```

Confusion Matrix

In [15]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test), ytic
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



Save The Model

In [16]:
```
feature_columns = X.columns.to_list()
joblib.dump(feature_columns,'AI_Impact_on_Jobs_2030.joblib')
```

Out[16]: ['AI_Impact_on_Jobs_2030.joblib']

In [17]:
```
joblib.dump(best_model,'bestmodel AI_Impact_on_Jobs_2030.joblib')
```

Out[17]: ['bestmodel AI_Impact_on_Jobs_2030.joblib']

If you have any suggestions, please DM me.

Even a small message from you can make a big impact on my career

I am Arun

In [ ]: