Hiring Decision predicting

Kanaka ratnam

Agenda

- Objective
  - Loading Data
  - Exploring Data
  - Encoding
  - Missing Values
  - Splitting the Data
  - Model Selection
- Model Training
  - Model Evaluation

objective

The objective of this predictive analysis can be multi-faceted, aiming to optimize and streamline the recruitment process, improve decision-making, and enhance overall hiring efficiency.

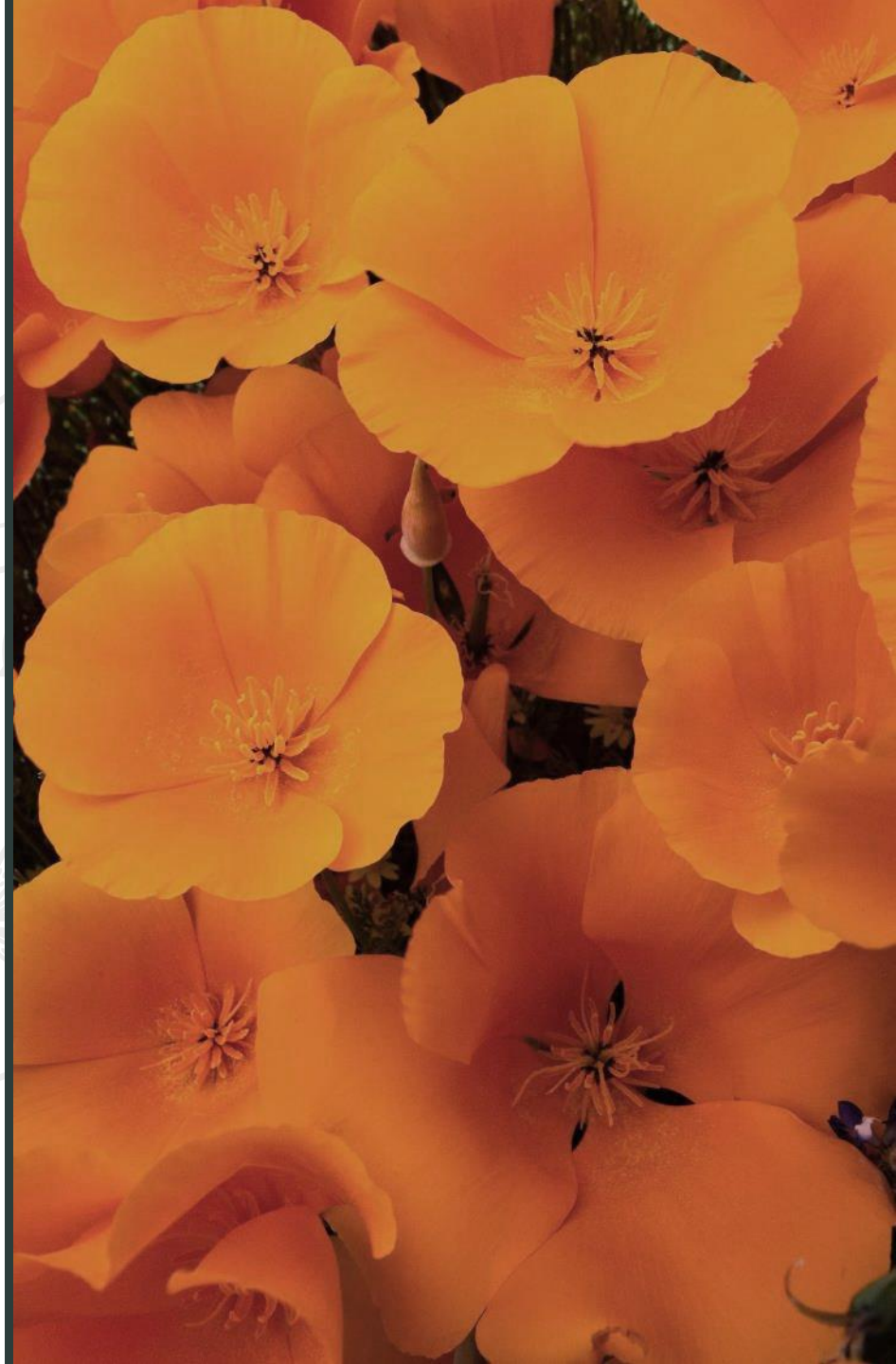
Here are some specific objectives:

1. Improve Hiring Efficiency.
2. Enhance Decision-Making.
3. Increase Hiring Success Rate.
4. Optimize Resource Allocation.
5. Improve Candidate Experience.
6. Enhance Diversity and Inclusion

Aim of the project

The goal is to predict whether a candidate will be hired based on these attributes. Here are the points considered as Aim of Predicting Hiring Decisions in Recruitment:

1. Improve Recruitment Efficiency.
2. Enhance Decision-Making.
3. Improve Candidate Experience.
4. Increase Hiring Quality.
5. Cost Reduction.



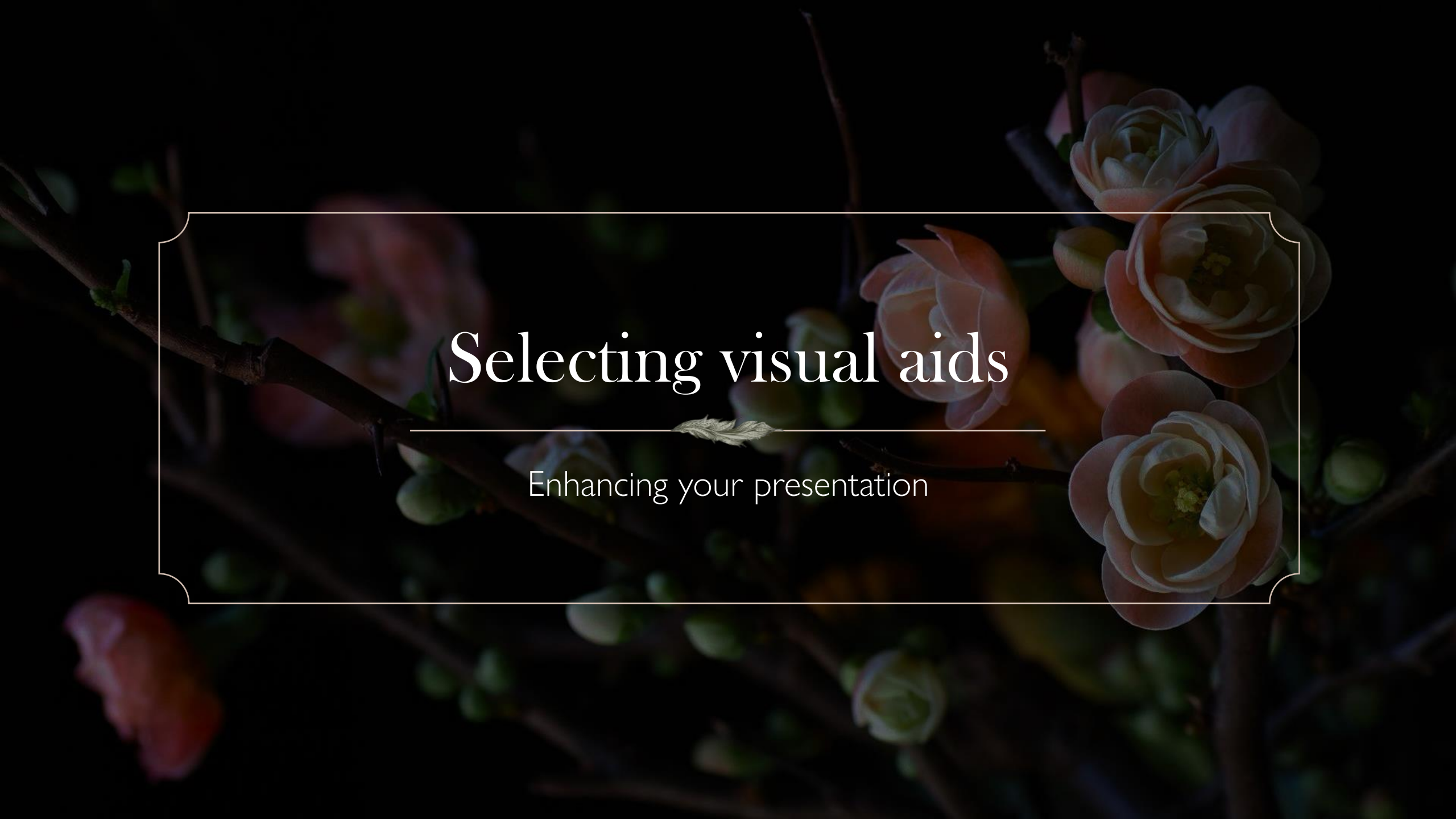
Loading Data

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df = pd.read_csv('/content/sample_data/recruitment_data.csv')
print("Data loaded successfully. Here's a preview:")
df.head()
```

```
Data loaded successfully. Here's a preview:
```

	Age	Gender	EducationLevel	ExperienceYears	PreviousCompanies	DistanceFromCompany	InterviewScore	SkillScore	PersonalityScore	RecruitmentStrategy	HiringDecision
0	26	1	2	0	3	26.783828	48	78	91	1	1
1	39	1	4	12	3	25.862694	35	68	80	2	1
2	48	0	2	3	2	9.920805	20	67	13	2	0
3	34	1	2	5	2	6.407751	36	27	70	3	0
4	30	0	1	6	1	43.105343	23	52	85	2	0

xt steps: Generate code with df | View recommended plots | New interactive sheet


Selecting visual aids
Enhancing your presentation

EXPLORING DATA

```
[3]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Age                  1500 non-null   int64  
 1   Gender               1500 non-null   int64  
 2   EducationLevel       1500 non-null   int64  
 3   ExperienceYears      1500 non-null   int64  
 4   PreviousCompanies    1500 non-null   int64  
 5   DistanceFromCompany  1500 non-null   float64
 6   InterviewScore       1500 non-null   int64  
 7   SkillScore           1500 non-null   int64  
 8   PersonalityScore     1500 non-null   int64  
 9   RecruitmentStrategy  1500 non-null   int64  
 10  HiringDecision       1500 non-null   int64  
dtypes: float64(1), int64(10)
memory usage: 129.0 KB
```

EXPLORING DATA

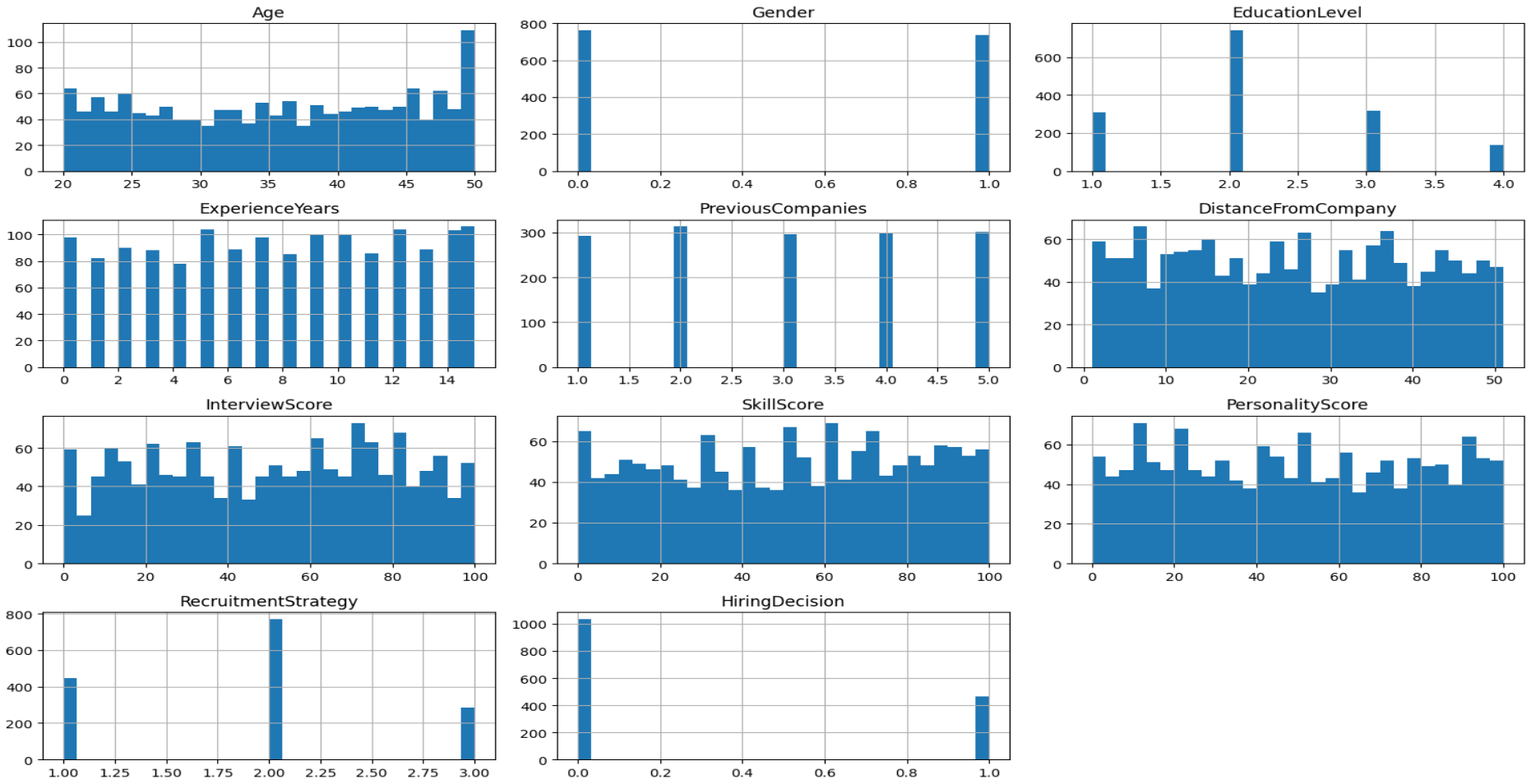
```
[ ] df.describe()
```

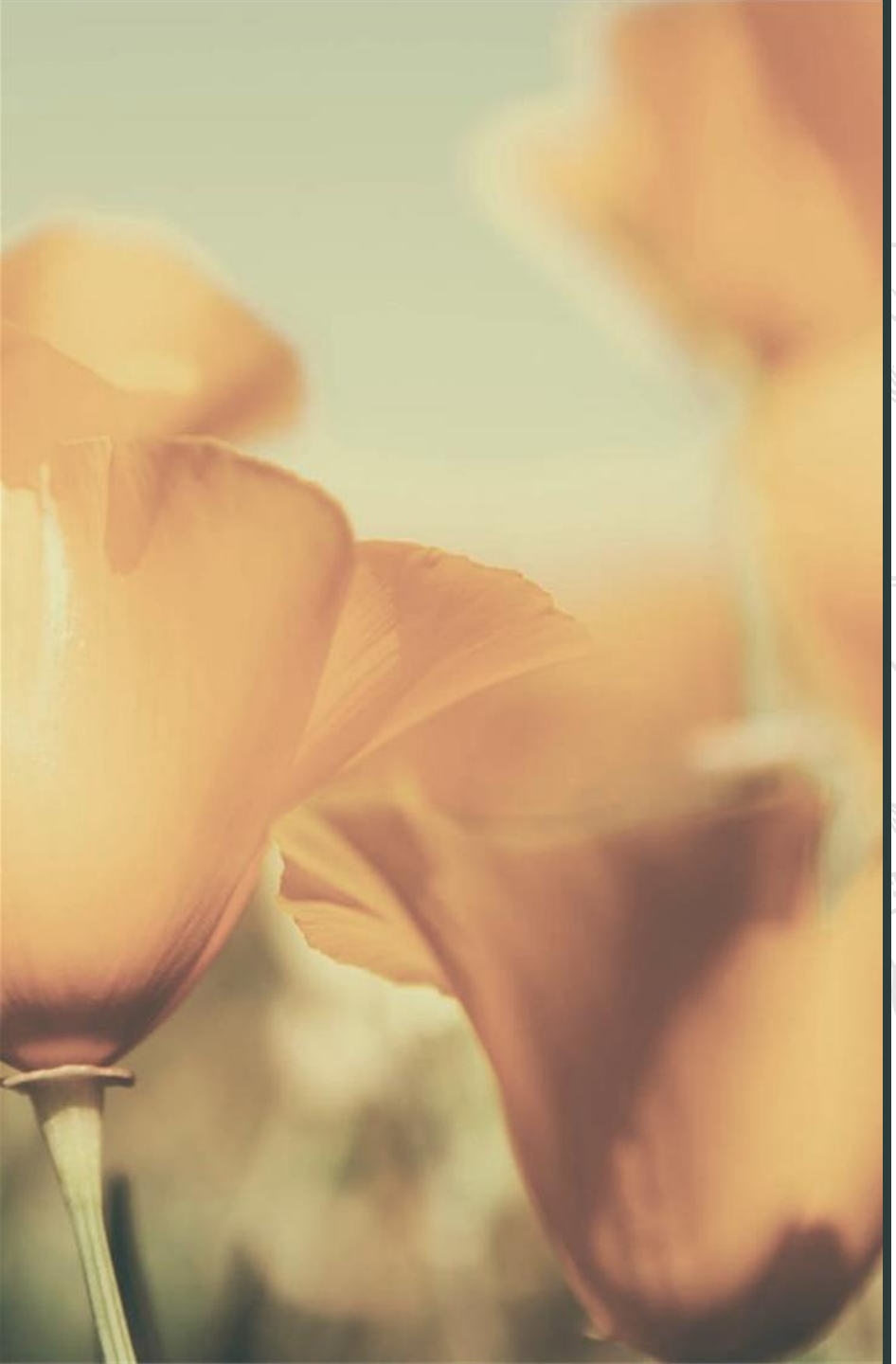
	Age	Gender	EducationLevel	ExperienceYears	PreviousCompanies	DistanceFromCompany	InterviewScore	SkillScore	PersonalityScore	RecruitmentStrategy	HiringDecisi
count	1500.000000	1500.000000	1500.000000	1500.000000	1500.00000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.0000
mean	35.148667	0.492000	2.188000	7.694000	3.00200	25.505379	50.564000	51.116000	49.387333	1.893333	0.3100
std	9.252728	0.500103	0.862449	4.641414	1.41067	14.567151	28.626215	29.353563	29.353201	0.689642	0.4626
min	20.000000	0.000000	1.000000	0.000000	1.00000	1.031376	0.000000	0.000000	0.000000	1.000000	0.0000
25%	27.000000	0.000000	2.000000	4.000000	2.00000	12.838851	25.000000	25.750000	23.000000	1.000000	0.0000
50%	35.000000	0.000000	2.000000	8.000000	3.00000	25.502239	52.000000	53.000000	49.000000	2.000000	0.0000
75%	43.000000	1.000000	3.000000	12.000000	4.00000	37.737996	75.000000	76.000000	76.000000	2.000000	1.0000
max	50.000000	1.000000	4.000000	15.000000	5.00000	50.992462	100.000000	100.000000	100.000000	3.000000	1.0000

```
[ ] df.columns
```

```
Index(['Age', 'Gender', 'EducationLevel', 'ExperienceYears',
       'PreviousCcompanies', 'DistanceFromCompany', 'InterviewScore',
       'SkillScore', 'PersonalityScore', 'RecruitmentStrategy',
       'HiringDecision'],
      dtype='object')
```

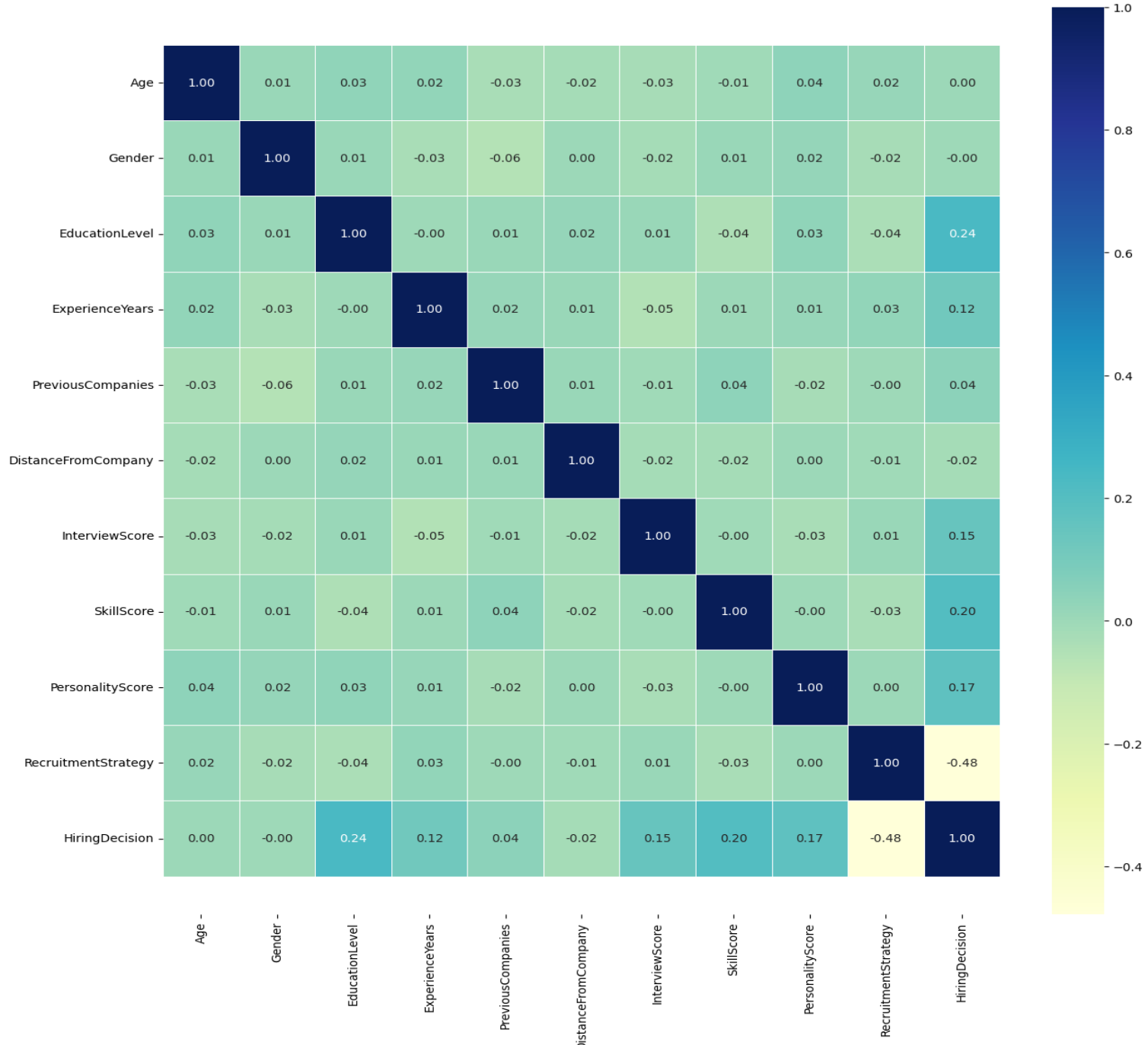
```
df.hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```





```
corr_matrix=df.select_dtypes([int,float]).corr()
print(corr_matrix)
```

```
                            Age    Gender  EducationLevel  ExperienceYears  \
Age                    1.000000  0.011286        0.032610         0.024780
Gender                 0.011286  1.000000        0.009676        -0.028502
EducationLevel         0.032610  0.009676        1.000000        -0.000951
ExperienceYears        0.024780 -0.028502       -0.000951         1.000000
PreviousCompanies     -0.032580 -0.061915        0.007367         0.015784
DistanceFromCompany   -0.021031  0.003822        0.020600         0.007920
InterviewScore        -0.025579 -0.020887        0.012807        -0.051701
SkillScore            -0.008068  0.012333       -0.043367         0.006323
PersonalityScore       0.037727  0.023729        0.031115         0.013891
RecruitmentStrategy    0.018064 -0.023753       -0.036923         0.029603
HiringDecision         0.001850 -0.002249        0.236710         0.122494

                      PreviousCompanies  DistanceFromCompany  InterviewScore  \
Age                           -0.032580            -0.021031       -0.025579
Gender                        -0.061915             0.003822       -0.020887
EducationLevel                 0.007367             0.020600        0.012807
ExperienceYears                0.015784             0.007920       -0.051701
PreviousCompanies              1.000000             0.009187       -0.008387
DistanceFromCompany            0.009187             1.000000       -0.019594
InterviewScore                -0.008387            -0.019594        1.000000
SkillScore                     0.040883            -0.016891       -0.004887
PersonalityScore              -0.024572             0.004627       -0.027967
RecruitmentStrategy           -0.000466            -0.007315        0.012004
HiringDecision                 0.044025            -0.016791        0.146064

                      SkillScore  PersonalityScore  RecruitmentStrategy  \
Age                    -0.008068          0.037727             0.018064
Gender                  0.012333          0.023729            -0.023753
EducationLevel         -0.043367          0.031115            -0.036923
ExperienceYears         0.006323          0.013891             0.029603
PreviousCompanies       0.040883         -0.024572            -0.000466
DistanceFromCompany    -0.016891          0.004627            -0.007315
InterviewScore         -0.004887         -0.027967             0.012004
SkillScore              1.000000         -0.004266            -0.031189
PersonalityScore       -0.004266          1.000000             0.004712
RecruitmentStrategy    -0.031189          0.004712             1.000000
HiringDecision          0.203668          0.169177            -0.477552

                      HiringDecision
Age                         0.001850
Gender                     -0.002249
EducationLevel              0.236710
ExperienceYears             0.122494
PreviousCompanies           0.044025
DistanceFromCompany        -0.016791
InterviewScore              0.146064
SkillScore                  0.203668
PersonalityScore            0.169177
```




```
[ ]  # Let's make our correlation matrix a little prettier
     corr_matrix = df.corr()
     fig, ax = plt.subplots(figsize=(15, 15))
     ax = sns.heatmap(corr_matrix,
                      annot=True,
                      linewidths=0.5,
                      fmt=".2f",
                      cmap="YlGnBu");
     bottom, top = ax.get_ylim()
     ax.set_ylim(bottom + 0.5, top - 0.5)
```

```
(11.5, -0.5)
```

ENCODING

```
[5] from sklearn.preprocessing import LabelEncoder
    obj_cols=df.select_dtypes(object).columns
    for col in obj_cols:
      encoder =LabelEncoder()
      df[col]=encoder.fit_transform(df[col])
    df.dtypes
```

	0
Age	int64
Gender	int64
EducationLevel	int64
ExperienceYears	int64
PreviousCompanies	int64
DistanceFromCompany	float64
InterviewScore	int64
SkillScore	int64
PersonalityScore	int64
RecruitmentStrategy	int64
HiringDecision	int64

dtype: object

Connected to

```
[6]  df.isnull().sum()
```

	0
Age	0
Gender	0
EducationLevel	0
ExperienceYears	0
PreviousCompanies	0
DistanceFromCompany	0
InterviewScore	0
SkillScore	0
PersonalityScore	0
RecruitmentStrategy	0
HiringDecision	0

dtype: int64

```
[7]  df.duplicated().sum()
```

```
0
```

MISSING VALUES

SPLITTING THE DATA

```
[8]  x=df.drop(columns=['HiringDecision'])
     y=df['HiringDecision']

     from sklearn.model_selection import train_test_split
     xtrain, xtest, ytrain, ytest = train_test_split(x,y,train_size=0.8)
```

```
[25] from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.svm import SVC
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
```

```
[14] log_reg = LogisticRegression()
     random_forest = RandomForestClassifier()
     svm = SVC()
     naive_bayes = MultinomialNB()
```

```python
models = {
    # Logistic Regression: Tune the regularization parameter C
    "Logistic Regression (C=0.5)": LogisticRegression(max_iter=1000, C=0.5),
    "Logistic Regression (C=1.0)": LogisticRegression(max_iter=1000, C=1.0),
    "Logistic Regression (C=2.0)": LogisticRegression(max_iter=1000, C=2.0),

    # Decision Tree: Tune depth of the tree
    "Decision Tree (depth=8)": DecisionTreeClassifier(max_depth=8, random_state=42),
    "Decision Tree (depth=12)": DecisionTreeClassifier(max_depth=12, random_state=42),
    "Decision Tree (depth=20)": DecisionTreeClassifier(max_depth=20, random_state=42),

    # Random Forest: Tune the number of trees (n_estimators) and maximum depth
    "Random Forest (100 estimators, depth=10)": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42),
    "Random Forest (200 estimators, depth=15)": RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42),
    "Random Forest (300 estimators, depth=20)": RandomForestClassifier(n_estimators=300, max_depth=20, random_state=42),

    # SVM: Tune the kernel and regularization parameter C
    "SVM (linear kernel, C=1.0)": SVC(kernel='linear', C=1.0),
    "SVM (linear kernel, C=2.0)": SVC(kernel='linear', C=2.0),
    "SVM (rbf kernel, C=1.0)": SVC(kernel='rbf', C=1.0),
    "SVM (rbf kernel, C=2.0)": SVC(kernel='rbf', C=2.0),

    # KNN: Tune the number of neighbors (n_neighbors)
    "KNN (k=3)": KNeighborsClassifier(n_neighbors=3),
    "KNN (k=5)": KNeighborsClassifier(n_neighbors=5),
    "KNN (k=7)": KNeighborsClassifier(n_neighbors=7)
}
```

Model Building

```python
#Initialize variables to store the best model information
best_model = None
best_model_name = ""
best_train_accuracy = 0
best_test_accuracy = 0

# Train and evaluate each model
for model_name, model in models.items():
    print(f"\nEvaluating {model_name}...\n")

    # Train the model
    model.fit(train, ytrain)

    # Predict on training data
    train_predictions = model.predict(xtrain)

    # Predict on testing data
    test_predictions = model.predict(xtest)

    # Evaluate the model
    train_accuracy = accuracy_score(ytrain, train_predictions) * 100
    test_accuracy = accuracy_score(ytest, test_predictions) * 100

    print(f"{model_name} Training Accuracy: {train_accuracy:.2f}%")
    print(f"{model_name} Testing Accuracy: {test_accuracy:.2f}%")

    print("Classification Report (Testing Data):")
    print(classification_report(ytest, test_predictions, labels=[-1, 0, 1], zero_division=0))

    # Update the best model if this model has the highest training accuracy
    if train_accuracy > best_train_accuracy:
        best_train_accuracy = train_accuracy
        best_model = model
        best_model_name = model_name

    # Track the highest testing accuracy
    if test_accuracy > best_test_accuracy:
        best_test_accuracy = test_accuracy
```

✓ Connected to Python 3 Google Compute Engine

Evaluating models for different values

Result for different models

```
Evaluating Logistic Regression (C=0.5)...

Logistic Regression (C=0.5) Training Accuracy: 85.42%
Logistic Regression (C=0.5) Testing Accuracy: 88.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.90      0.94      0.92       205
           1       0.85      0.77      0.81        95

   micro avg       0.88      0.88      0.88       300
   macro avg       0.58      0.57      0.57       300
weighted avg       0.88      0.88      0.88       300


Evaluating Logistic Regression (C=1.0)...

Logistic Regression (C=1.0) Training Accuracy: 85.50%
Logistic Regression (C=1.0) Testing Accuracy: 88.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.90      0.94      0.92       205
           1       0.85      0.77      0.81        95

   micro avg       0.88      0.88      0.88       300
   macro avg       0.58      0.57      0.57       300
weighted avg       0.88      0.88      0.88       300
```

```
Evaluating Logistic Regression (C=2.0)...

Logistic Regression (C=2.0) Training Accuracy: 85.58%
Logistic Regression (C=2.0) Testing Accuracy: 88.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.90      0.94      0.92       205
           1       0.85      0.77      0.81        95

   micro avg       0.88      0.88      0.88       300
   macro avg       0.58      0.57      0.57       300
weighted avg       0.88      0.88      0.88       300


Evaluating Decision Tree (depth=8)...

Decision Tree (depth=8) Training Accuracy: 98.08%
Decision Tree (depth=8) Testing Accuracy: 93.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.96      0.95       205
           1       0.90      0.87      0.89        95

   micro avg       0.93      0.93      0.93       300
   macro avg       0.61      0.61      0.61       300
weighted avg       0.93      0.93      0.93       300
```

```
Evaluating Decision Tree (depth=12)...

Decision Tree (depth=12) Training Accuracy: 99.92%
Decision Tree (depth=12) Testing Accuracy: 91.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.93      0.93       205
           1       0.85      0.86      0.86        95

   micro avg       0.91      0.91      0.91       300
   macro avg       0.60      0.60      0.60       300
weighted avg       0.91      0.91      0.91       300


Evaluating Decision Tree (depth=20)...

Decision Tree (depth=20) Training Accuracy: 100.00%
Decision Tree (depth=20) Testing Accuracy: 90.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.92      0.93       205
           1       0.83      0.87      0.85        95

   micro avg       0.90      0.90      0.90       300
   macro avg       0.59      0.60      0.59       300
weighted avg       0.91      0.90      0.90       300
```

Result for different models

```
Evaluating Decision Tree (depth=20)...

Decision Tree (depth=20) Training Accuracy: 100.00%
Decision Tree (depth=20) Testing Accuracy: 90.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.92      0.93       205
           1       0.83      0.87      0.85        95

   micro avg       0.90      0.90      0.90       300
   macro avg       0.59      0.60      0.59       300
weighted avg       0.91      0.90      0.90       300


Evaluating Random Forest (100 estimators, depth=10)...

Random Forest (100 estimators, depth=10) Training Accuracy: 99.50%
Random Forest (100 estimators, depth=10) Testing Accuracy: 94.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.98      0.96       205
           1       0.94      0.87      0.91        95

   micro avg       0.94      0.94      0.94       300
   macro avg       0.63      0.62      0.62       300
weighted avg       0.94      0.94      0.94       300
```

```
Evaluating Random Forest (200 estimators, depth=15)...

Random Forest (200 estimators, depth=15) Training Accuracy: 100.00%
Random Forest (200 estimators, depth=15) Testing Accuracy: 94.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.94      0.98      0.96       205
           1       0.94      0.87      0.91        95

   micro avg       0.94      0.94      0.94       300
   macro avg       0.63      0.62      0.62       300
weighted avg       0.94      0.94      0.94       300


Evaluating Random Forest (300 estimators, depth=20)...

Random Forest (300 estimators, depth=20) Training Accuracy: 100.00%
Random Forest (300 estimators, depth=20) Testing Accuracy: 95.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.95      0.98      0.96       205
           1       0.95      0.88      0.92        95

   micro avg       0.95      0.95      0.95       300
   macro avg       0.63      0.62      0.63       300
weighted avg       0.95      0.95      0.95       300
```

```
Evaluating SVM (linear kernel, C=1.0)...

SVM (linear kernel, C=1.0) Training Accuracy: 86.67%
SVM (linear kernel, C=1.0) Testing Accuracy: 88.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.89      0.95      0.92       205
           1       0.87      0.75      0.80        95

   micro avg       0.88      0.88      0.88       300
   macro avg       0.59      0.56      0.57       300
weighted avg       0.88      0.88      0.88       300


Evaluating SVM (linear kernel, C=2.0)...

SVM (linear kernel, C=2.0) Training Accuracy: 86.58%
SVM (linear kernel, C=2.0) Testing Accuracy: 88.33%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.89      0.95      0.92       205
           1       0.87      0.75      0.80        95

   micro avg       0.88      0.88      0.88       300
   macro avg       0.59      0.56      0.57       300
weighted avg       0.88      0.88      0.88       300
```

Result for different models

```
Evaluating SVM (rbf kernel, C=1.0)...

SVM (rbf kernel, C=1.0) Training Accuracy: 72.00%
SVM (rbf kernel, C=1.0) Testing Accuracy: 69.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.70      0.97      0.81       205
           1       0.57      0.08      0.15        95

   micro avg       0.69      0.69      0.69       300
   macro avg       0.42      0.35      0.32       300
weighted avg       0.66      0.69      0.60       300


Evaluating SVM (rbf kernel, C=2.0)...

SVM (rbf kernel, C=2.0) Training Accuracy: 72.17%
SVM (rbf kernel, C=2.0) Testing Accuracy: 68.67%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.70      0.96      0.81       205
           1       0.53      0.09      0.16        95

   micro avg       0.69      0.69      0.69       300
   macro avg       0.41      0.35      0.32       300
weighted avg       0.64      0.69      0.60       300
```

```
Evaluating KNN (k=3)...

KNN (k=3) Training Accuracy: 80.50%
KNN (k=3) Testing Accuracy: 64.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.71      0.80      0.75       205
           1       0.41      0.31      0.35        95

   micro avg       0.64      0.64      0.64       300
   macro avg       0.37      0.37      0.37       300
weighted avg       0.62      0.64      0.62       300


Evaluating KNN (k=5)...

KNN (k=5) Training Accuracy: 77.83%
KNN (k=5) Testing Accuracy: 68.00%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.72      0.86      0.79       205
           1       0.49      0.29      0.37        95

   micro avg       0.68      0.68      0.68       300
   macro avg       0.41      0.38      0.38       300
weighted avg       0.65      0.68      0.65       300
```

```
Evaluating KNN (k=7)...

KNN (k=7) Training Accuracy: 76.08%
KNN (k=7) Testing Accuracy: 67.67%
Classification Report (Testing Data):
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.72      0.85      0.78       205
           1       0.48      0.31      0.37        95

   micro avg       0.68      0.68      0.68       300
   macro avg       0.40      0.38      0.39       300
weighted avg       0.65      0.68      0.65       300
```

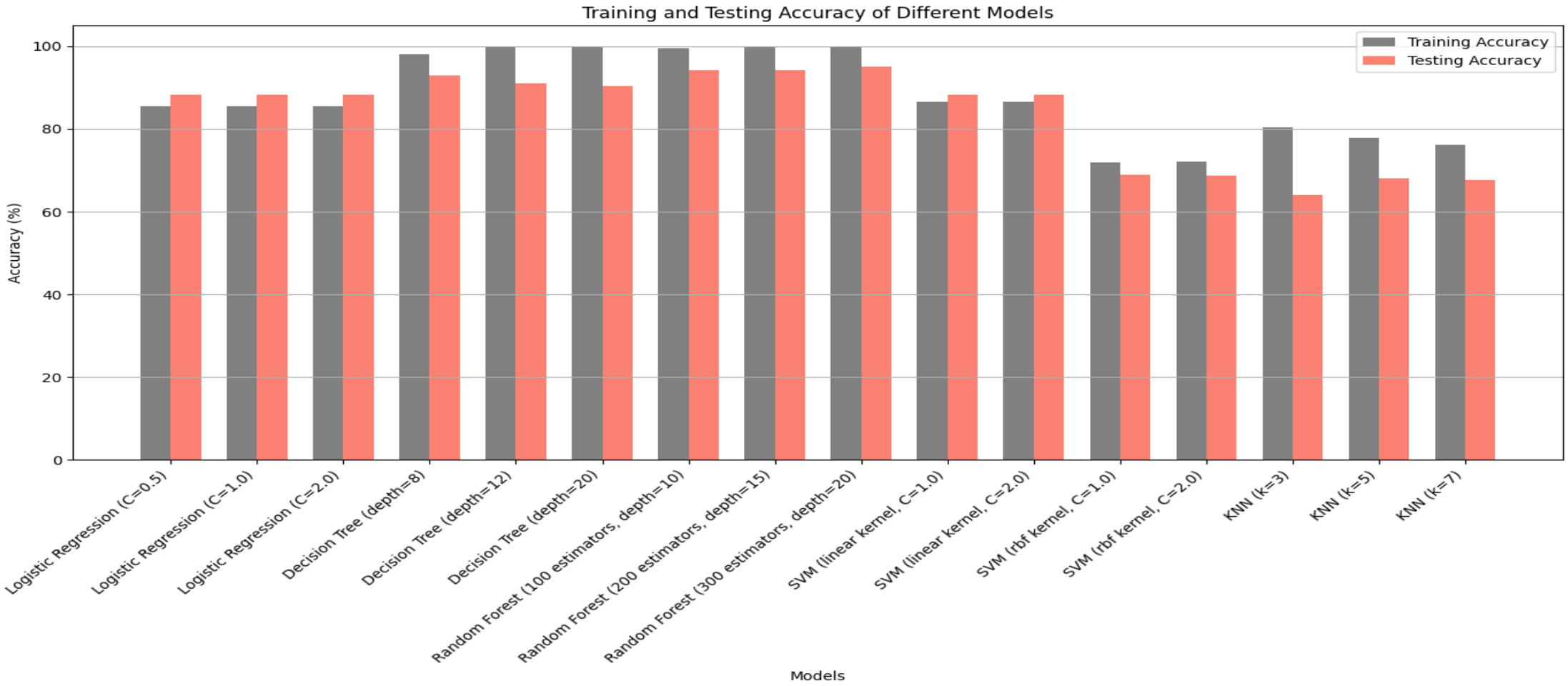
```
[31] # Define a threshold for acceptable testing accuracy
     acceptable_test_accuracy_threshold = 0.9 * best_test_accuracy

     # Initialize variables to store accuracies and model names
     model_names = []
     train_accuracies = []
     test_accuracies = []

     # Collect the accuracies and model names from the previous code
     for model_name, model in models.items():
         # Train the model
         model.fit(xtrain, ytrain)

         # Predict on training and testing data
         train_predictions = model.predict(xtrain)
         test_predictions = model.predict(xtest)

         # Calculate accuracies
         train_accuracy = accuracy_score(ytrain, train_predictions) * 100
         test_accuracy = accuracy_score(ytest, test_predictions) * 100

         # Store the accuracies and model names
         model_names.append(model_name)
         train_accuracies.append(train_accuracy)
         test_accuracies.append(test_accuracy)
```

Training the data

```
2] # Plot the training and testing accuracies for different models
   plt.figure(figsize=(14, 8))
   x = range(len(model_names))

   # Bar width
   bar_width = 0.35

   # Plotting bars
   plt.bar(x, train_accuracies, width=bar_width, label='Training Accuracy', align='center', color='grey')
   plt.bar([i + bar_width for i in x], test_accuracies, width=bar_width, label='Testing Accuracy', align='center', color='salmon')

   # Adding labels and title
   plt.xlabel('Models')
   plt.ylabel('Accuracy (%)')
   plt.title('Training and Testing Accuracy of Different Models')
   plt.xticks([i + bar_width / 2 for i in x], model_names, rotation=45, ha='right')
   plt.legend()
   plt.grid(axis='y')

   # Display the plot
   plt.tight_layout()
   plt.show()
```

Training and Testing Accuracy for different values

Graph of training and testing accuracy values



```python
# Final evaluation and prediction using the best model
final_test_accuracy = accuracy_score(ytest, best_model.predict(xtest)) * 100

if final_test_accuracy >= acceptable_test_accuracy_threshold:
    print(f"\nFinal Evaluation of the Best Model ({best_model_name}):\n")
    final_predictions = best_model.predict(xtest)
    train_predictions = best_model.predict(xtrain)

    print(f"Best Model Training Accuracy: {accuracy_score(ytrain, train_predictions) * 100:.2f}%")
    print(f"Best Model Testing Accuracy: {accuracy_score(ytest, final_predictions) * 100:.2f}%")
    print("Classification Report:")
    print(classification_report(ytest, final_predictions, labels=[ 0, 1], zero_division=0))
else:
    print("The model with the highest training accuracy does not meet the acceptable testing accuracy threshold.")
```

```
Final Evaluation of the Best Model (Decision Tree (depth=20)):

Best Model Training Accuracy: 100.00%
Best Model Testing Accuracy: 89.33%
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.93       209
           1       0.87      0.76      0.81        91

    accuracy                           0.89       300
   macro avg       0.89      0.86      0.87       300
weighted avg       0.89      0.89      0.89       300
```

Model Evaluation

Conclusion

The model performs well overall, with an accuracy of 88%, indicating that it is capable of correctly predicting hiring decisions for a significant majority of the candidates.

- The model shows a high precision (0.90) and recall (0.95) for Class 0 (Not Hired), which suggests that it is very effective at identifying candidates who should not be hired. For Class 1 (Hired), the precision is lower at 0.87, the recall is at 0.76. This indicates that while the model is good at identifying most of the candidates who should be hired, it has a higher rate of false positives compared to Class 0.
- The macro average and weighted average scores are close to each other and reflect balanced performance across both classes. This is a positive indicator that the model is not heavily biased towards one class over the other.

Thank you

Kanaka ratnam