CAPSTONE PROJECT

PROJECT TITLE

Presented By:
Vani Sai Deepika – RISHI MS Institute of Engineering & Technology for Women – Computer Science and Engineering

OUTLINE

- Problem Statement
- System Development Approach
- Algorithm & Deployment
- Result
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

- With the growing emphasis on fairness and transparency in hiring, organizations need intelligent systems to estimate employee salaries based on job roles and personal qualifications.
- In today's digital era, HR departments collect massive volumes of employee data but often rely on manual analysis and outdated rules for salary decisions.
- This project aims to develop a predictive model using structured demographic and employment data to classify whether an individual's income exceeds ₹50,000 per year.
- Key attributes such as age, education level, occupation, marital status, hours worked per week, and work class are used to uncover patterns in historical salary records.
- By automating the salary estimation process, the project supports smarter, faster, and fairer decision-making in HR operations.
- Additionally, it lays the groundwork for building scalable, data-driven HR analytics tools in future workforce ecosystems.

SYSTEM APPROACH

System requirements:

- Python
- Jupyter Notebook / VS Code
- Streamlit
- Streamlit Cloud
- GitHub

Library required to build the model:

- Pandas
- Numpy
- scikit-learn (sklearn)
- Joblib
- Matplotlib
- Seaborn
- streamlit

Machine Learning Models Used :

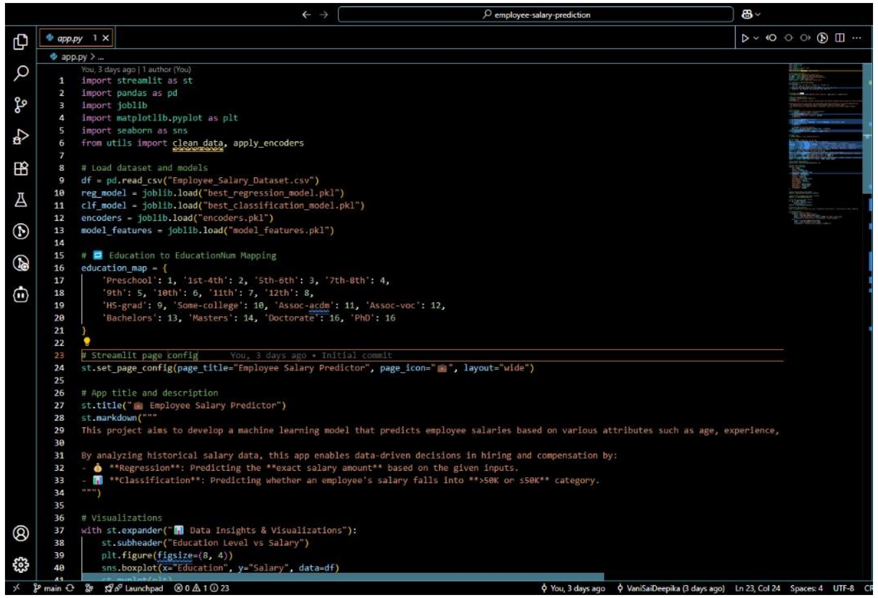
- Linear Regression
- Random Forest Regressor (Regression Model)
- Random Forest Classifier (Classification Model)

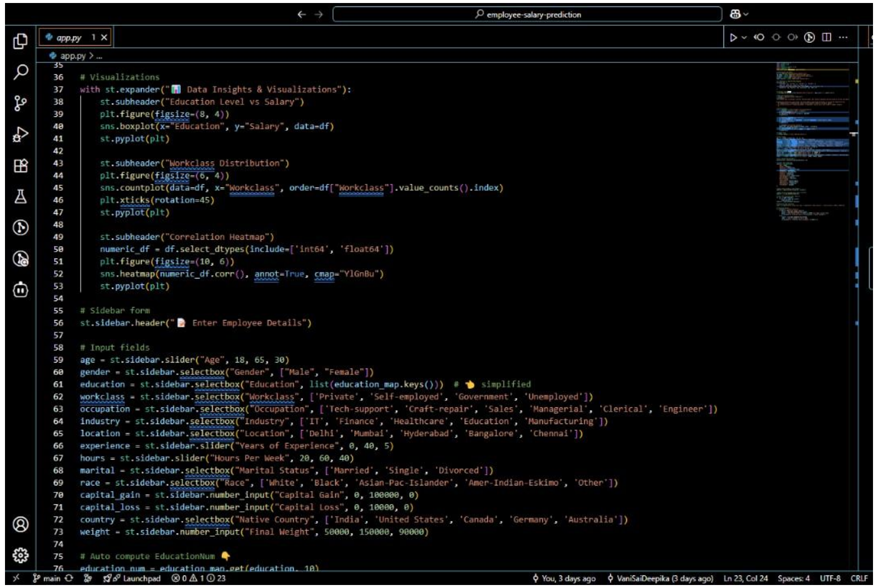
ALGORITHM & DEPLOYMENT

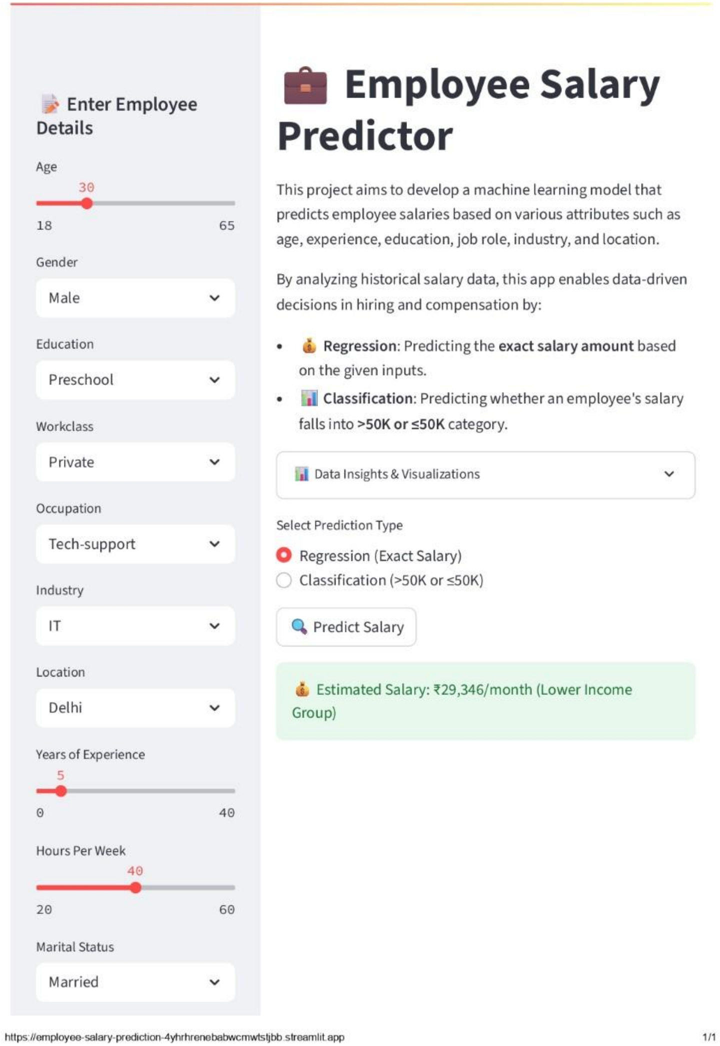
- In this project, I implemented both **regression and classification algorithms** to predict employee salaries. For regression tasks, I used **Linear Regression** and **Random Forest Regressor** to estimate the exact salary value. For classification, I applied a **Random Forest Classifier** to determine whether a person's income is **greater than ₹50K or not**.
- The process began with **data loading and exploration**, where I imported the dataset from a CSV file using pandas and performed initial analysis using seaborn and matplotlib to understand feature distributions and correlations.
- Next, in the **preprocessing stage**, I handled missing values and encoded categorical variables using **Label Encoding** and **One-Hot Encoding** techniques.
- Then came **feature selection and engineering**, where I chose relevant features such as age, education, workclass, occupation, and hours per week, and applied normalization or encoding as required.
- For **model building**, I trained the **Linear Regression** and **Random Forest Regressor** for predicting salary amounts and the **Random Forest Classifier** for predicting income category.
- The models were evaluated using appropriate metrics: **RMSE** and **R^2 Score** for regression, and **Accuracy** and **Confusion Matrix** for classification.
- After evaluation, I used joblib to **save the trained models and encoders** for use during deployment.
- I then built an intuitive **web interface using Streamlit**, where users could enter input data through a form and get instant predictions.
- Finally, the application was **deployed via Streamlit Cloud**, making it publicly accessible through a GitHub-connected deployment.

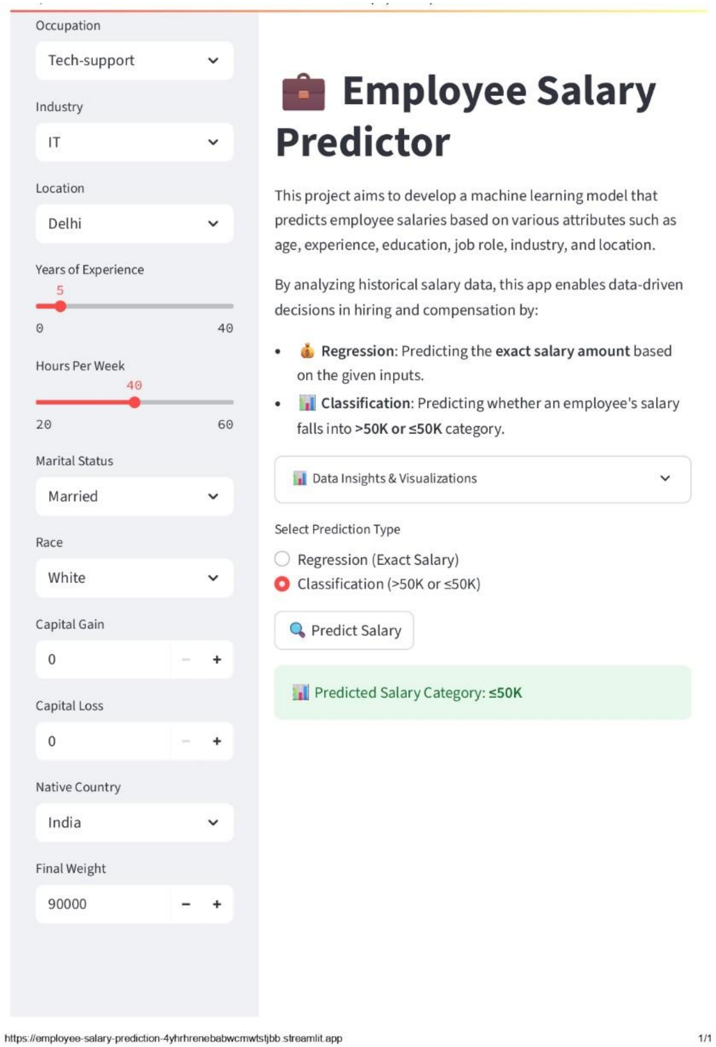
RESULT

Model Output & App Interface



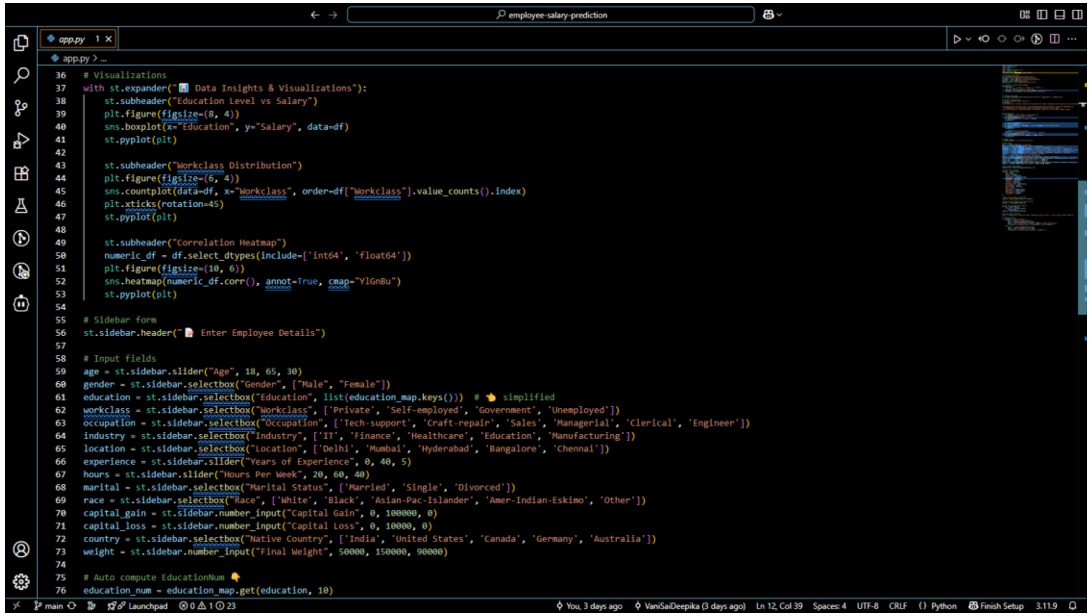


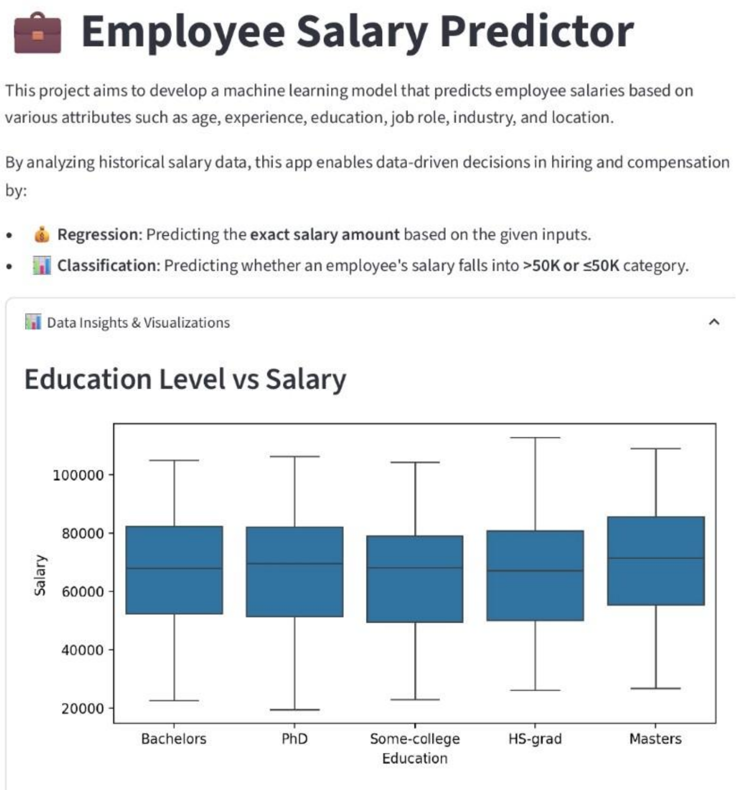


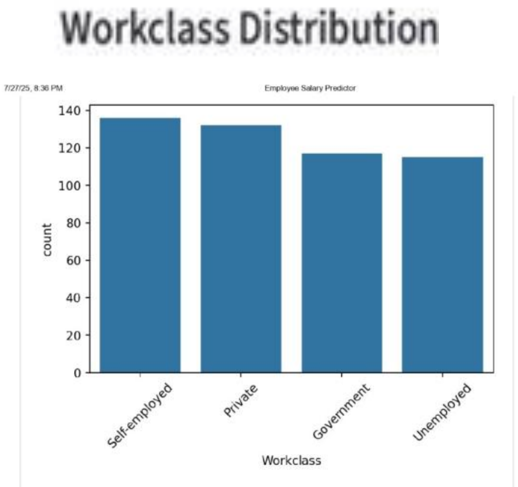


edunet foundation

RESULT







edunet foundation

RESULT

Regression & Classification Insights

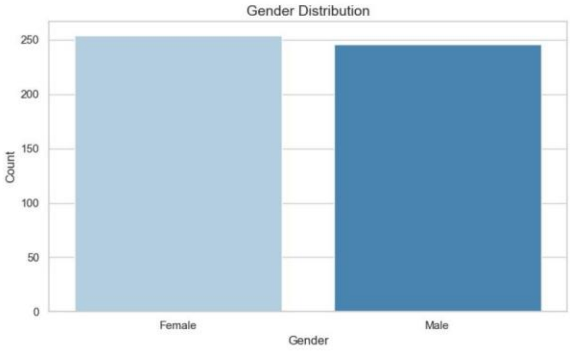
```python
# salary_model_dev.ipynb

# Step 1: Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor,
RandomForestClassifier
from sklearn.metrics import (
    mean_absolute_error, mean_squared_error, r2_score,
    accuracy_score, confusion_matrix, classification_report
)
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")  # Suppress warnings
sns.set(style="whitegrid")

# Step 2: Load Dataset
df = pd.read_csv("Employee_Salary_Dataset.csv")

# Step 3: EDA
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Gender', palette="Blues")
plt.title("Gender Distribution", fontsize=14)
plt.xlabel("Gender")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 5))
sns.histplot(df['Salary'], kde=True, color="green")
plt.title("Salary Distribution", fontsize=14)
plt.xlabel("Salary")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# Top 10 Highest Earning Professions
if 'Job_Role' in df.columns:
    top_jobs = df.groupby('Job_Role')
['Salary'].mean().sort_values(ascending=False).head(10)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_jobs.values, y=top_jobs.index, palette="mako")
    plt.title("Top 10 Highest Earning Professions", fontsize=14)
    plt.xlabel("Average Salary")
```

```python
    plt.ylabel("Job Role")
    plt.tight_layout()
    plt.show()

# Step 4: Preprocessing
df.dropna(inplace=True)

# Binary encode Gender
df['Gender'] = LabelEncoder().fit_transform(df['Gender'])

# Rename experience column if needed
if "Experience_Years" in df.columns:
    df.rename(columns={"Experience_Years": "YearsExperience"},
inplace=True)

# Drop ID column if it exists
if "ID" in df.columns:
    df.drop("ID", axis=1, inplace=True)

# One-hot encode all categorical columns (excluding targets)
cat_cols = df.select_dtypes(include='object').columns.tolist()
if "Salary_Class" in cat_cols:
    cat_cols.remove("Salary_Class")
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# Save final feature names for later
all_feature_columns = df.drop("Salary", axis=1).columns.tolist()

# Step 5A: REGRESSION SETUP
X_reg = df.drop("Salary", axis=1)
y_reg = df["Salary"]
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X_reg,
y_reg, test_size=0.2, random_state=42)

# Step 6A: Regression Models
lr = LinearRegression()
rf = RandomForestRegressor(random_state=42)

lr.fit(X_train_r, y_train_r)
rf.fit(X_train_r, y_train_r)

# Step 7A: Evaluate Regression
def evaluate_reg(name, y_true, y_pred):
    print(f"\n\U0001F4CC {name} Regression Metrics:")
    print("MAE:", round(mean_absolute_error(y_true, y_pred), 2))
    print("MSE:", round(mean_squared_error(y_true, y_pred), 2))
    print("R2 Score:", round(r2_score(y_true, y_pred), 4))
    print("-" * 40)
```

```python
evaluate_reg("Linear Regression", y_test_r, lr.predict(X_test_r))
evaluate_reg("Random Forest", y_test_r, rf.predict(X_test_r))

# Step 8A: Feature Importance Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=rf.feature_importances_, y=X_reg.columns,
palette="viridis")
plt.title("Random Forest Regressor - Feature Importances",
fontsize=14)
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

# Step 9A: Save Best Regression Model
best_reg_model = rf if r2_score(y_test_r, rf.predict(X_test_r)) >
r2_score(y_test_r, lr.predict(X_test_r)) else lr
best_reg_model.feature_names_in_ = np.array(all_feature_columns)
joblib.dump(best_reg_model, "best_regression_model.pkl")
print("\u2705 Regression model saved as 'best_regression_model.pkl'")

# Step 5B: CLASSIFICATION SETUP
df['Salary_Class'] = (df['Salary'] > 50000).astype(int)
X_clf = df.drop(["Salary", "Salary_Class"], axis=1)
y_clf = df["Salary_Class"]
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_clf,
y_clf, test_size=0.2, random_state=42)

# Step 6B: Classification Models
logreg = LogisticRegression(max_iter=1000)
rfc = RandomForestClassifier(random_state=42)

logreg.fit(X_train_c, y_train_c)
rfc.fit(X_train_c, y_train_c)

# Step 7B: Evaluate Classification
def evaluate_clf(name, y_true, y_pred):
    print(f"\n\U0001F4CC {name} Classification Metrics:")
    print("Accuracy:", round(accuracy_score(y_true, y_pred), 4))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true,
y_pred))
    print("-" * 40)

evaluate_clf("Logistic Regression", y_test_c,
logreg.predict(X_test_c))
evaluate_clf("Random Forest Classifier", y_test_c,
rfc.predict(X_test_c))
```

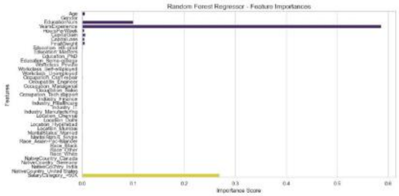
```python
# Step 9B: Save Best Classification Model
best_clf_model = rfc if accuracy_score(y_test_c,
rfc.predict(X_test_c)) > accuracy_score(y_test_c,
logreg.predict(X_test_c)) else logreg
best_clf_model.feature_names_in_ = np.array(X_clf.columns)
joblib.dump(best_clf_model, "best_classification_model.pkl")
print("\u2705 Classification model saved as
'best_classification_model.pkl'")

# Step 10: Correlation Heatmap (Improved)
plt.figure(figsize=(14, 10))
corr_matrix = df.corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(
    corr_matrix,
    mask=mask,
    cmap=cmap,
    vmin=-1,
    vmax=1,
    center=0,
    square=True,
    linewidths=0.5,
    annot=True,
    fmt=".2f",
    annot_kws={"size": 8}
)
plt.title("\U0001F517 Feature Correlation Heatmap", fontsize=16,
fontweight="bold")
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# Step 11: Save Final Model Features for Input Alignment
joblib.dump(all_feature_columns, "model_features.pkl")
print("□ Feature columns saved as 'model_features.pkl'")

# Step 12: Save Label Encoders for applicable columns
from sklearn.preprocessing import LabelEncoder

# Create encoders only for label-encoded columns
label_columns = ['Gender']
encoders = {}
for col in label_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le
```

```python
joblib.dump(encoders, "encoders.pkl")
print("□ Encoders saved as 'encoders.pkl'")
```



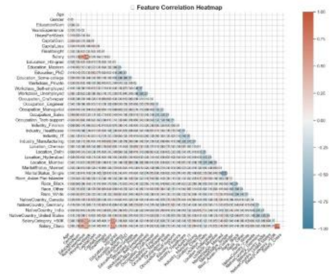

```
□ Linear Regression Regression Metrics:
MAE: 3694.83
MSE: 18767567.81
R2 Score: 0.9469
```

```
----------------------------------------

□ Random Forest Regression Metrics:
MAE: 4150.05
MSE: 25808668.41
R2 Score: 0.9269
----------------------------------------
```




```
□ Regression model saved as 'best_regression_model.pkl'

□ Logistic Regression Classification Metrics:
Accuracy: 0.99
Confusion Matrix:
 [[17  1]
 [ 0 82]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        18
           1       0.99      1.00      0.99        82

    accuracy                           0.99       100
   macro avg       0.99      0.97      0.98       100
weighted avg       0.99      0.99      0.99       100

----------------------------------------

□ Random Forest Classifier Classification Metrics:
Accuracy: 1.0
Confusion Matrix:
 [[18  0]
 [ 0 82]]
Classification Report:
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        82

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100

----------------------------------------
□ Classification model saved as 'best_classification_model.pkl'
```




```
□ Feature columns saved as 'model_features.pkl'
□ Encoders saved as 'encoders.pkl'
```

edunet foundation

RESULT

- Git hub link: https://github.com/VANISAIDEEPIKA/Employee-Salary-Prediction.git

CONCLUSION

- This project effectively demonstrates how **machine learning can be applied to real-world HR analytics**, particularly in predicting employee salaries. The model supports both:
- **Regression** – to predict the actual salary amount
- **Classification** – to predict whether salary is > ₹50,000 or not
- By incorporating both techniques, the tool becomes **flexible**, addressing varied analytical needs in hiring, workforce planning, and compensation benchmarking.
- The **Streamlit web app** enhances usability by offering a clean, form-based interface that allows users to input employee data and instantly view predictions

Challenges Faced & Resolutions:

- **Challenge Faced:**

Handling diverse categorical features like gender, education, and work class

- **Resolution:**

Used Label Encoding and One-Hot Encoding, with encoders saved using joblib for consistent deployment.

- **Challenge Faced:**

Mismatch between training features and real-time prediction input

- **Resolution:**

Built a robust preprocessing pipeline in utils.py to mirror the exact transformations used during training.

- **Challenge Faced:**

Difficulty in saving and reloading ML models within the Streamlit environment

- **Resolution:**

Followed a modular approach using helper functions to load .pkl models cleanly at runtime.

edunet foundation

FUTURE SCOPE

- Integrate advanced deep learning models for comparison.
- Include additional inputs like job role, certifications, or performance scores.
- Enable real-time integration with HRMS platforms for automated salary insights.
- Incorporate NLP to analyze resumes or job descriptions for richer predictions.
- Convert the Streamlit app into a responsive Progressive Web App (PWA) for mobile HR use cases

REFERENCES

- scikit-learn documentation – https://scikit-learn.org/stable/
- Streamlit documentation – https://docs.streamlit.io/
- pandas documentation – https://pandas.pydata.org/docs/
- NumPy documentation – https://numpy.org/doc/
- Matplotlib documentation – https://matplotlib.org/stable/contents.html
- Seaborn documentation – https://seaborn.pydata.org/
- joblib documentation – https://joblib.readthedocs.io/en/latest/
- Streamlit Community Cloud Docs – https://docs.streamlit.io/streamlit-community-cloud

THANK YOU