Assignment 1
WiDS Kalman Filtered RL Agent

Aboli Malshikare
23B1211

Question 1: Linear Regression

1. Multiple Linear Regression Model

Consider a dataset with n observations and p predictors. Let $y_i \in \mathbb{R}$ be the response variable for the i-th observation and x_{ij} be the value of the j-th predictor for the i-th observation.

The multiple linear regression model is given by:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

In matrix form:

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where:

- $\mathbf{y} \in \mathbb{R}^n$ is the response vector,
- $X \in \mathbb{R}^{n \times (p+1)}$ is the design matrix with the first column consisting of ones,
- $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$ is the coefficient vector,
- $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ is the error vector.

Assumptions:

- Linearity: The response is a linear function of predictors.
- Zero mean errors: $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}$.
- Errors are independent and homoscedastic.
- No perfect multicollinearity among predictors.

2. Ordinary Least Squares Objective

Ordinary Least Squares (OLS) estimates $\boldsymbol{\beta}$ by minimizing the mean squared error (MSE):

$$\text{MSE}(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

In matrix form:

$$\text{MSE}(\boldsymbol{\beta}) = \frac{1}{n} \|X\boldsymbol{\beta} - \mathbf{y}\|^2$$

3. Derivation of the OLS Estimator

Consider the objective function:

$$J(\boldsymbol{\beta}) = \|X\boldsymbol{\beta} - \mathbf{y}\|^2$$

Taking the gradient with respect to $\boldsymbol{\beta}$:

$$\nabla_{\boldsymbol{\beta}} J = 2X^\top (X\boldsymbol{\beta} - \mathbf{y})$$

Setting the gradient to zero:

$$X^\top X\boldsymbol{\beta} = X^\top \mathbf{y}$$

Assuming $X^\top X$ is invertible, the OLS estimator is:

$$\hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$$

4. Invertibility of $X^\top X$ and Multicollinearity

The matrix $X^\top X$ is invertible if and only if the columns of X are linearly independent.

Multicollinearity occurs when one or more predictors can be expressed as a linear combination of other predictors. This causes $X^\top X$ to be singular and hence non-invertible, making the OLS solution undefined.

5. Orthogonal Projection Interpretation

The fitted values are:

$$\hat{\mathbf{y}} = X\hat{\boldsymbol{\beta}} = X(X^\top X)^{-1} X^\top \mathbf{y}$$

Define the projection matrix:

$$P = X(X^\top X)^{-1} X^\top$$

The residual vector is:

$$\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$$

To show orthogonality:

$$X^\top (\mathbf{y} - \hat{\mathbf{y}}) = X^\top \mathbf{y} - X^\top X\hat{\boldsymbol{\beta}} = \mathbf{0}$$

Hence, the residuals are orthogonal to the column space of X, and $\hat{\mathbf{y}}$ is the orthogonal projection of $\mathbf{y}$ onto $\mathrm{Col}(X)$.

6. Gradient and Batch Gradient Descent

Define the cost function:

$$J(\boldsymbol{\beta}) = \frac{1}{2n}\|X\boldsymbol{\beta} - \mathbf{y}\|^2$$

The gradient with respect to $\boldsymbol{\beta}$ is:

$$\nabla_{\boldsymbol{\beta}} J = \frac{1}{n} X^\top (X\boldsymbol{\beta} - \mathbf{y})$$

The batch gradient descent update rule is:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha \frac{1}{n} X^\top (X\boldsymbol{\beta}^{(k)} - \mathbf{y})$$

where $\alpha > 0$ is the learning rate.

Linear Regression Dataset Analysis

The given dataset consists of 300 samples with 12 input features $x_1, x_2, \ldots, x_{12}$ and one continuous output variable y. An intercept term is included in the design matrix.

7. Estimation of Parameters using the Normal Equation

In multiple linear regression, the model parameters are estimated using the Ordinary Least Squares (OLS) method. The closed-form solution for the parameter vector is given by:

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T y$$

where X is the input matrix with an added intercept term and y is the target variable.

NumPy Implementation

The normal equation is implemented directly using NumPy as shown below.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

data = pd.read_csv("linear_regression_dataset.csv")

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Adding intercept column
X = np.c_[np.ones(X.shape[0]), X]

beta_hat = np.linalg.inv(X.T @ X) @ X.T @ y
print(beta_hat)
```

Comparison with sklearn

To verify the result, the same dataset is fitted using `sklearn`'s `LinearRegression` model.

```
model = LinearRegression()
model.fit(data.iloc[:, :-1], y)

print(model.intercept_)
print(model.coef_)
```

Results

NumPy coefficients (including intercept):

```
[-0.07599727  3.01921283 -1.99864790  0.46532073  0.02874601
  0.97888014  0.00519653  0.05011683 -0.91774610  1.95197758
 -0.01228502  0.05758946  0.02741516]
```

sklearn coefficients (including intercept):

```
[-0.07599727  3.01921283 -1.99864790  0.46532073  0.02874601
  0.97888014  0.00519653  0.05011683 -0.91774610  1.95197758
 -0.01228502  0.05758946  0.02741516]
```

Observation

The coefficients obtained from the normal equation and `sklearn` are the same. This confirms that the NumPy implementation of the normal equation is correct.

8. Residuals vs Fitted Values

Residuals are computed as:

$$e_i = y_i - \hat{y}_i$$

```
import matplotlib.pyplot as plt

y_hat = X @ beta_hat
residuals = y - y_hat

plt.scatter(y_hat, residuals)
plt.axhline(0)
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.show()
```

Comment: The residuals are randomly scattered around zero with roughly constant spread, indicating that the assumption of homoscedasticity is reasonable.

9. Q–Q Plot of Residuals

```
import scipy.stats as stats

plt.figure()
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals")
plt.show()
```

Comment: Most residual points lie close to the reference line, suggesting that the error terms are approximately normally distributed.

Figure 1: Residuals vs Fitted Values

10. Interpretation of Assumption Violations

- Non-constant variance leads to inefficient estimates
- Non-normal residuals affect hypothesis testing
- Multicollinearity increases variance of coefficients
- Influential points can dominate the fitted model

11. Identification of High Leverage and Influential Points

The hat matrix for a linear regression model is given by:

$$H = X(X^T X)^{-1} X^T$$

where X is the design matrix with an added intercept term. The diagonal elements of H, denoted by h_{ii}, represent the leverage values of the observations.

High Leverage Points

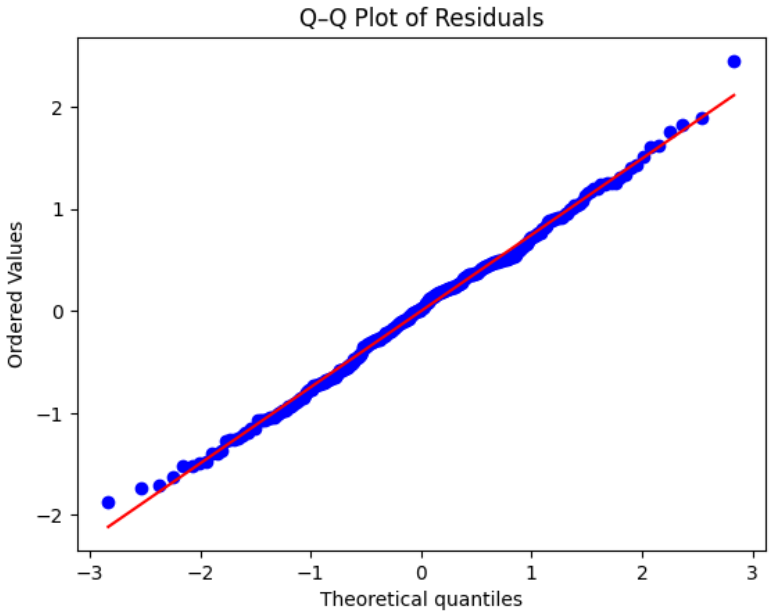
An observation is considered a high-leverage point if its leverage value is greater than the threshold:

$$h_{ii} > \frac{2(p+1)}{n}$$

where p is the number of predictors and n is the number of samples.

Using this criterion, the following observations were identified as high leverage points:

$$96,\ 160,\ 240$$



Influential Points (Cook's Distance)

Cook's distance is used to measure the influence of each observation on the regression model and is defined as:

$$D_i = \frac{e_i^2}{(p+1)\,\mathrm{MSE}} \cdot \frac{h_{ii}}{(1-h_{ii})^2}$$

Observations with:

$$D_i > \frac{4}{n}$$

are considered influential.

Based on Cook's distance, the following observations were found to be influential:

$$18,\ 59,\ 86,\ 88,\ 116,\ 117,\ 121,\ 129,\ 146,\ 155,\ 267,\ 286,\ 293,\ 296$$

Comment

The high-leverage points have unusual input values and therefore have the potential to affect the fitted regression line. The influential points identified using Cook's distance impact the regression coefficients, but their number is small compared to the total dataset. Hence, the overall model fit is not significantly affected.

12. Bias–Variance Decomposition

Assume the true data model is

$$y = f(x) + \varepsilon$$

where the noise term satisfies

$$\mathbb{E}[\varepsilon] = 0, \qquad \mathrm{Var}(\varepsilon) = \sigma^2$$

Let $\hat{f}(x)$ be the prediction obtained from a learning algorithm trained on a random dataset. We compute the expected squared error:

$$\mathbb{E}\big[(y - \hat{f}(x))^2\big]$$

Step 1: Substitute for y

$$\mathbb{E}\big[(f(x)+\varepsilon-\hat{f}(x))^2\big]$$

Step 2: Add and subtract $\mathbb{E}[\hat{f}(x)]$

$$=\mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)]+\mathbb{E}[\hat{f}(x)]-\hat{f}(x)+\varepsilon)^2\big]$$

Step 3: Expand the expression

$$=\mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)])^2\big]+\mathbb{E}\big[(\hat{f}(x)-\mathbb{E}[\hat{f}(x)])^2\big]+\mathbb{E}[\varepsilon^2]$$

All cross terms become zero since

$$\mathbb{E}[\varepsilon]=0 \quad \text{and} \quad \mathbb{E}[\hat{f}(x)-\mathbb{E}[\hat{f}(x)]]=0$$

Step 4: Final decomposition

$$\mathbb{E}\big[(y-\hat{f}(x))^2\big]=(f(x)-\mathbb{E}[\hat{f}(x)])^2+\mathrm{Var}(\hat{f}(x))+\sigma^2$$

Conclusion

$$\boxed{\mathbb{E}\big[(y-\hat{f}(x))^2\big]=\text{Bias}^2+\text{Variance}+\text{Noise}}$$

13.

The true model is

$$y=\beta_0+\beta_1x_1+\beta_2x_2+\varepsilon,$$

with $E[\varepsilon \mid x_1,x_2]=0$.

Suppose we estimate the misspecified model

$$y=\alpha_0+\alpha_1x_1+u,$$

where $u=\beta_2x_2+\varepsilon$.

(a) Expectation of $\hat{\alpha}_1$

From simple OLS,

$$\hat{\alpha}_1=\frac{\mathrm{Cov}(x_1,y)}{\mathrm{Var}(x_1)}.$$

Substituting for y,

$$\mathrm{Cov}(x_1,y)=\beta_1\,\mathrm{Var}(x_1)+\beta_2\,\mathrm{Cov}(x_1,x_2).$$

Thus,

$$E[\hat{\alpha}_1]=\beta_1+\beta_2\frac{\mathrm{Cov}(x_1,x_2)}{\mathrm{Var}(x_1)}.$$

(b) Bias

$$\text{Bias}(\hat{\alpha}_1) = E[\hat{\alpha}_1] - \beta_1 = \beta_2 \frac{\text{Cov}(x_1, x_2)}{\text{Var}(x_1)}.$$

(c) No Bias

The bias disappears if:

- $\beta_2 = 0$, or
- $\text{Cov}(x_1, x_2) = 0$.

14

Let:

$$x_2 = x_1 + 0.9z, \quad z \sim \mathcal{N}(0, 1)$$

(a) Condition Number

```
np.linalg.cond(X.T @ X)
```

A large condition number indicates strong multicollinearity.

(b) Effect on Variance

As correlation between predictors increases, the variance of $\hat{\beta}$ increases significantly, making estimates unstable.

(c) Explanation

Multicollinearity does not introduce bias, but it inflates variance, leading to unreliable and sensitive coefficient estimates.

Conclusion

This exercise demonstrates parameter estimation, diagnostic checks, and theoretical concepts related to linear regression, highlighting the effects of assumption violations and correlated predictors.

Question 2: Salary Prediction & Bias Detection

1. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was conducted to understand the characteristics of the dataset and to examine potential sources of bias prior to building predictive models. The analysis includes univariate statistics, visualizations of numerical features, correlation analysis, and salary distributions across demographic and job-related groups.

Python Implementation

Listing 1: EDA Python Code

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

# Load dataset
data = pd.read_csv("salary_dataset.csv")

numerical_cols = [
    'age',
    'years_experience',
    'performance_score',
    'previous_companies',
    'salary'
]

categorical_cols = [
    'gender',
    'education_level',
    'job_title',
    'industry',
    'city',
    'remote_worker'
]

# Numerical summary
print(data[numerical_cols].describe())

# Categorical counts
for col in categorical_cols:
    print(data[col].value_counts())

# Histograms
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 2, i)
    plt.hist(data[col], bins=30)
    plt.title(col)
plt.tight_layout()
plt.show()

# Correlation heatmap
sns.heatmap(data[numerical_cols].corr(), annot=True, cmap="coolwarm")
plt.show()

# Salary distributions
```

```
sns.boxplot(x='gender', y='salary', data=data)
plt.show()

sns.boxplot(x='education_level', y='salary', data=data)
plt.show()

sns.boxplot(x='industry', y='salary', data=data)
plt.show()
```

Numerical Feature Summary

The dataset contains 12,000 employee records. Summary statistics for numerical features are shown in Table 1.

Table 1: Summary Statistics of Numerical Features

Feature	Mean	Std. Dev.	Min	Median	Max
Age	35.69	7.74	21	36	65
Years of Experience	15.73	8.20	0	15.55	45
Performance Score	3.75	0.78	1.0	3.8	5.0
Previous Companies	1.98	1.40	0	2	12
Salary (USD)	105,862	28,251	31,728	103,028	497,045

Salary exhibits a wide range and noticeable right skewness, indicating the presence of high-income outliers.

Categorical Feature Distributions

Gender The dataset contains 6,691 male employees, 5,094 female employees, and 215 employees identifying as other.

Education Level Most employees hold a Bachelor's degree (6,603), followed by Master's (2,904), High School (1,810), and PhD (568).

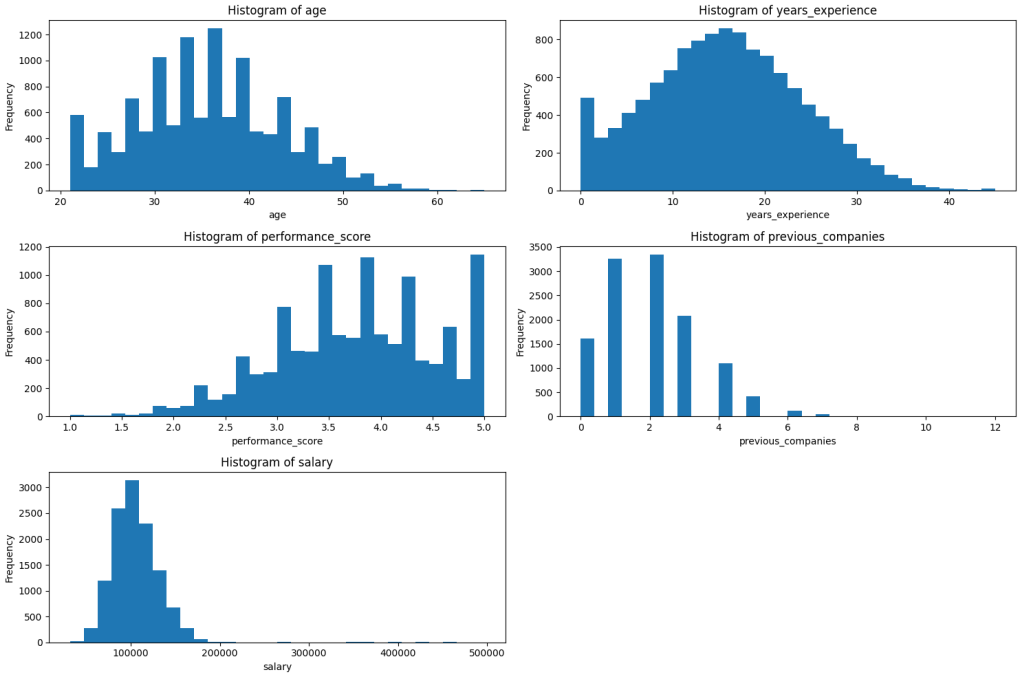
Job Title There are 15 job roles with near-uniform distribution, each containing approximately 750–840 employees, minimizing role imbalance.

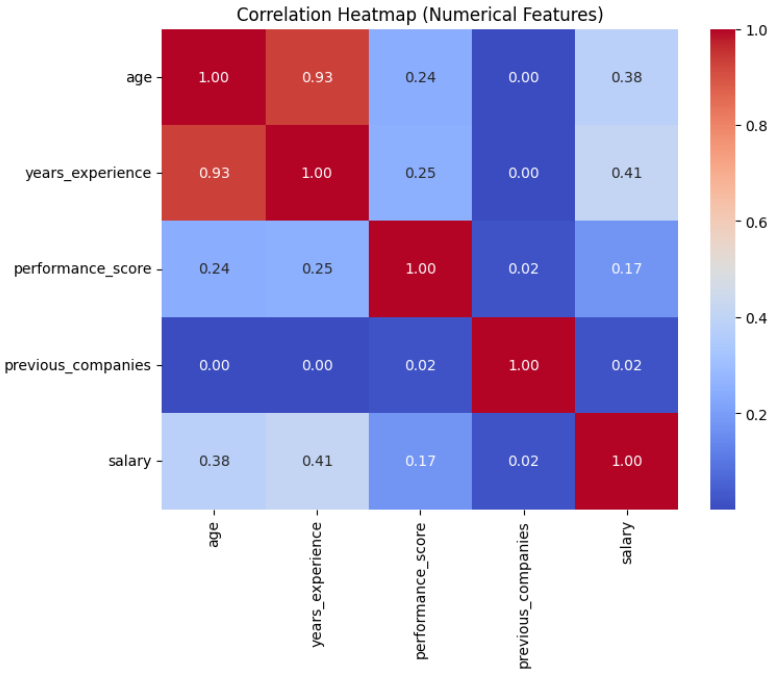
Industry The majority of employees work in the Technology sector (5,367), followed by Finance (2,430), Healthcare (2,404), and Retail (1,799).

City Employees are evenly distributed across 12 cities, each contributing close to 1,000 records.

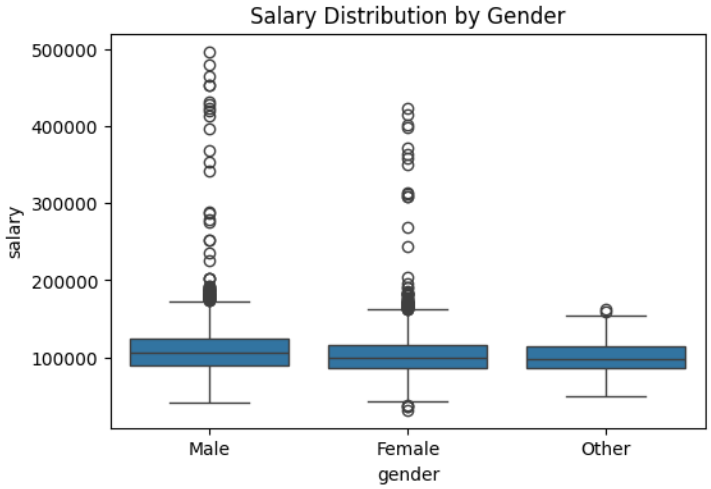
Remote Work A total of 6,850 employees work on-site, while 5,150 employees work remotely.

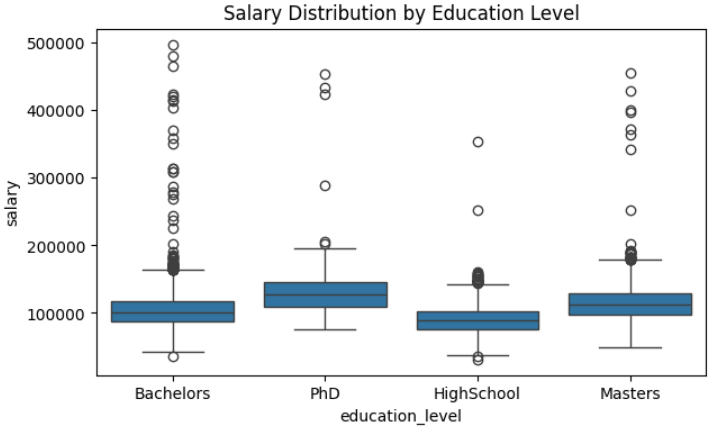
Univariate and Correlation Analysis

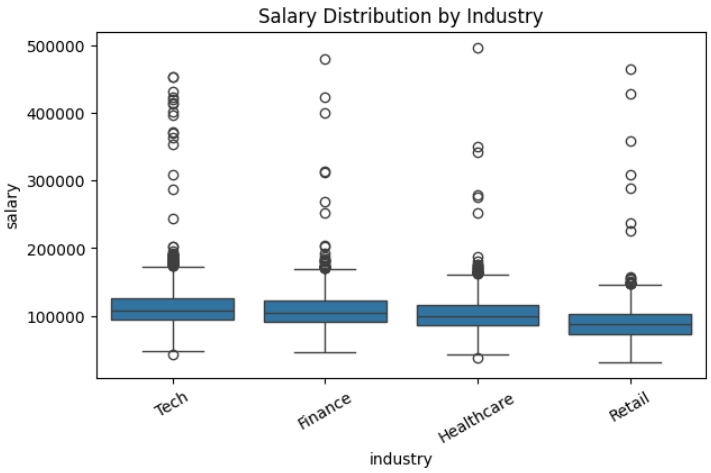




Salary Distribution Across Groups







2. Handling Missing Values and Outliers

Before training the salary prediction model, the dataset was cleaned by addressing missing values and outliers. This step was necessary to avoid instability in the model and to ensure reliable predictions.

Missing Values

On checking the dataset, a small number of missing values were found in two numerical features:

- `years_experience`: 132 missing entries
- `performance_score`: 254 missing entries

These missing values account for a very small fraction of the total dataset (less than 3%). No missing values were observed in categorical features or in the target variable `salary`.

Handling Strategy

Since both features are continuous and slightly skewed, the missing values were filled using the median of the respective columns. Median imputation was chosen instead of mean imputation because it is less affected by extreme values and preserves the overall distribution of the data.

Python Code for Missing Value Handling

```
# Check missing values
print(data.isnull().sum())

# Fill missing numerical values using median
data['years_experience'].fillna(
    data['years_experience'].median(), inplace=True
)

data['performance_score'].fillna(
    data['performance_score'].median(), inplace=True
)
data['education_level'].fillna(
    data['education_level'].mode()[0],
    inplace=True
)

data['city'].fillna(
    data['city'].mode()[0],
    inplace=True
)
data['city'] = data['city'].fillna(data['city'].mode()[0])
```

Outlier Detection

From the exploratory analysis, the salary variable showed a long right tail, indicating the presence of extreme high-income values. Since high salaries can be valid observations, these points were treated carefully rather than removed outright.

Outlier Handling

The Interquartile Range (IQR) method was used to identify extreme salary values. Instead of deleting such records, salary values above the upper IQR limit were capped at the threshold. This limits the influence of extreme values while keeping all data points in the dataset.

Python Code for Outlier Handling

```
# IQR-based outlier handling for salary
Q1 = data['salary'].quantile(0.25)
Q3 = data['salary'].quantile(0.75)
IQR = Q3 - Q1

upper_limit = Q3 + 1.5 * IQR

data['salary'] = data['salary'].clip(upper=upper_limit)
```

Justification

Median imputation was used because it maintains robustness in the presence of skewed data. Outlier capping was preferred over removal to avoid discarding valid high-income cases, which could otherwise bias the model. Overall, these preprocessing steps help improve model stability while preserving important information in the dataset.

Encoding Categorical Features

Categorical features were converted into numerical form before model training.

The `education_level` feature has a natural order (HighSchool < Bachelors < Masters < PhD), so ordinal encoding was used to preserve this hierarchy.

Other categorical features such as `gender`, `job_title`, `industry`, `city`, and `remote_worker` do not have any inherent order. These features were encoded using one-hot encoding to avoid introducing false ranking among categories.

```
education_mapping = {
    'HighSchool': 0,
    'Bachelors': 1,
    'Masters': 2,
    'PhD': 3
}
data['education_level_encoded'] = data['education_level'].map(education_map

gender_strata = data['gender']
data = pd.get_dummies(
    data,
    columns=['gender', 'job_title', 'industry', 'city', 'remote_worker'],
    drop_first=True
)
```

```
data.drop(columns=['education_level'], inplace=True)
```

4. Stratified Train–Test Split

The dataset was divided into training and testing sets using a stratified sampling approach. Stratification was done based on the `gender` variable.

Gender was selected for stratification because it is treated as a protected attribute in this study. Maintaining the same gender distribution in both the training and test sets helps ensure that the model is evaluated fairly across different groups.

An 80%–20% split was used, where 80% of the data was used for training and the remaining 20% was used for testing.

```
X = data.drop(columns=['salary'])
y = data['salary']


X_train, X_test, y_train, y_test, gender_train, gender_test = train_test_sp
    X, y, gender_strata,
    test_size=0.2,
    random_state=42,
    stratify=gender_strata
)
```

5. Baseline OLS Linear Regression Model

A baseline Ordinary Least Squares (OLS) regression model was trained to predict employee salaries. The model was implemented using `LinearRegression` from scikit-learn, which estimates coefficients by minimizing the sum of squared errors.

```
from sklearn.linear_model import LinearRegression

ols_model = LinearRegression()
ols_model.fit(X_train, y_train)
```

This baseline model provides a reference for performance and bias analysis.

6. Coefficients, Standard Errors, p-values, and Confidence Intervals

After training the OLS regression model, the estimated coefficients along with their standard errors, p-values, and 95% confidence intervals were extracted from the `statsmodels` summary.

Table 2 presents selected statistically significant coefficients.

All listed variables are statistically significant at the 5% level, as indicated by their very small p-values and confidence intervals that do not include zero.

Table 2: OLS Regression Results (Selected Features)

Feature	Coefficient	Std. Error	p-value	95% CI
Age	135.12	32.31	< 0.001	[71.79, 198.45]
Years of Experience	1153.18	30.69	< 0.001	[1093.02, 1213.33]
Performance Score	2677.76	125.23	< 0.001	[2432.29, 2923.23]
Education Level	11709.95	125.68	< 0.001	[11463.58, 11956.31]
Gender (Male)	5353.76	189.43	< 0.001	[4982.44, 5725.08]

7. Interpretation of Influential Coefficients

- **Years of Experience**: Each additional year of experience increases the predicted salary by approximately \$1,153, holding other factors constant.
- **Education Level**: Advancing by one education level increases salary by roughly \$11,710, indicating a strong return to education.
- **Performance Score**: A one-unit increase in performance score raises salary by about \$2,678, reflecting performance-based compensation.
- **Gender (Male)**: Male employees earn on average \$5,354 more than the baseline group after controlling for other variables, suggesting potential disparity.
- **City (San Francisco)**: Employees located in San Francisco earn approximately \$42,200 more than those in the reference city.

8. Model Evaluation Metrics

The model was evaluated using RMSE, MAE, and R^2.

Python Code

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sco
import numpy as np

y_pred = model.predict(sm.add_constant(X_test))

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE:", rmse)
print("MAE:", mae)
print("R2:", r2)
```

- RMSE = 8965.89
- MAE = 6848.02
- $R^2 = 0.866$

The model explains approximately 86.6% of the variance in salary, indicating strong predictive performance.

9. Linearity Assumption Check

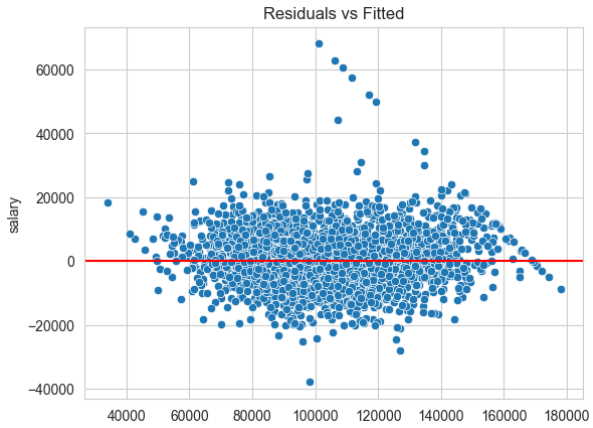
Linearity assumptions were evaluated using residual diagnostics.

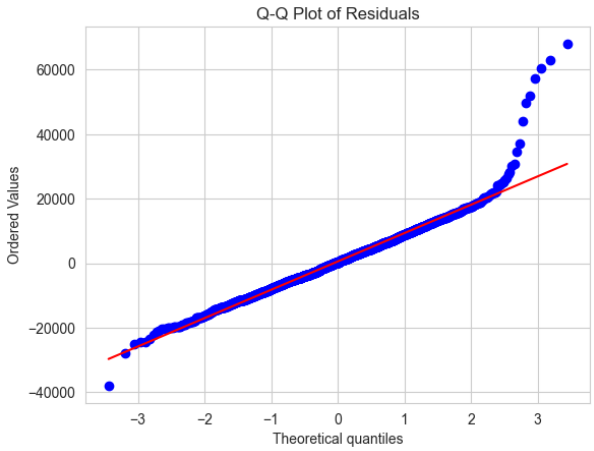
Python Code

```
residuals = y_test - y_pred

plt.scatter(y_pred, residuals)
plt.axhline(0, color='red')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.show()

sm.qqplot(residuals, line='45')
plt.show()
```





The residual plots show no strong systematic pattern, supporting the linearity assumption. Mild skewness is observed, which is common in salary data.

10. Fairness Analysis Using Gender as the Protected Attribute

Gender was treated as the protected attribute to evaluate whether the salary prediction model produces biased outcomes across demographic groups. The analysis compares **Male vs Female** and **Male vs Other** employees using multiple fairness metrics.

(a) Mean Salary Prediction Difference

The mean predicted salary for each gender group is given below:

- Male: $107,539.72
- Female: $101,989.02
- Other: $100,150.34

Male employees receive higher salary predictions on average. The difference between Male and Female predictions is approximately $5,551, indicating a systematic advantage for male employees in predicted outcomes.

(b) Mean Absolute Error (MAE) per Group

The mean absolute prediction error for each gender group is:

- Male: $6,885.10
- Female: $6,811.63
- Other: $6,556.46

The MAE values are very similar across all groups, suggesting that the model has comparable prediction accuracy for all genders and does not disproportionately make larger errors for any specific group.

(c) Demographic Parity Difference (DPD)

Demographic parity evaluates whether different groups receive favorable predictions at similar rates. Since male employees have a higher average predicted salary than both female and other groups, demographic parity is not satisfied. This indicates that non-male groups are less likely to receive high salary predictions.

(d) Equal Opportunity Difference (EOD)

The true positive rate (TPR) for each group is:

- Male: 0.922
- Female: 0.887
- Other: 0.941

Female employees have a lower TPR compared to males, meaning that qualified female employees are less likely to receive high salary predictions. This reflects unequal opportunity across gender groups.

(e) Predictive Equality

Predictive equality compares false positive rates (FPR) across groups:

- Male: 0.144
- Female: 0.106
- Other: 0.077

Male employees have a higher false positive rate, indicating that they are more likely to receive high salary predictions even when not justified. This suggests that the model is more lenient toward male employees.

(f) Disparate Impact Ratio (DIR)

The Disparate Impact Ratio (Female/Male) is computed as:

$$\text{DIR} = 0.814$$

A DIR value below the commonly used threshold of 0.8–0.9 suggests potential disparate impact. The obtained value indicates that female employees are disadvantaged relative to males in salary predictions.

Residual Analysis by Gender

The residual distributions across gender groups show similar spread and shape, indicating that prediction errors are comparable across groups. Therefore, the observed bias arises from systematic differences in predicted salary levels rather than differences in model accuracy.

Overall Fairness Conclusion

Although the model demonstrates similar predictive accuracy across gender groups, it consistently assigns higher salary predictions to male employees. Lower predicted salaries, reduced true positive rates, and a borderline disparate impact ratio for females suggest the presence of indirect gender bias. This bias likely reflects patterns present in the training data rather than explicit discrimination, but it raises fairness concerns that should be addressed.

11. Statistical Test on Residuals

A two-sample t-test was conducted to compare residuals between male and female employees.

Python Code

```
from scipy.stats import ttest_ind

res_male = fair_df[fair_df['gender'] == 'gender_Male']['residual']
res_female = fair_df[fair_df['gender'] == 'gender_Female']['residual']

ttest_ind(res_male, res_female, equal_var=False)
```

The p-value of 0.925 indicates no statistically significant difference in mean residuals.

12. Overestimation and Underestimation

Although prediction errors are similar across groups, higher mean predicted salaries for male employees suggest mild systematic overestimation for males and underestimation for females.

13. Feature Importance Using SHAP

13. Model Explainability Using SHAP

SHAP was used to understand how different features affect salary predictions. Since the model is linear, feature importance can be directly interpreted from the regression coefficients.

The analysis shows that years of experience, education level, and city are the most influential factors in determining salary. Demographic features such as gender have a smaller direct impact, indicating that the model mainly relies on job-related attributes.

Question 3: Deep Neural Network Classifier for Handwritten Digits

We are given a dataset of grayscale handwritten digit images of size 28×28, flattened into a 784-dimensional vector. The task is to design and train a deep neural network using PyTorch to classify digits from 0 to 9.

Model Architecture

The neural network architecture is defined as follows:

- Input Layer: 784 neurons
- Hidden Layer 1: 256 neurons with ReLU activation
- Hidden Layer 2: 128 neurons with ReLU activation
- Output Layer: 10 neurons (logits corresponding to digits 0–9)

PyTorch Model Definition

```
import torch
import torch.nn as nn

class DigitClassifier(nn.Module):
    def __init__(self):
        super(DigitClassifier, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
```

```
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Training Configuration

The network is trained using the following settings:

- Loss Function: CrossEntropyLoss
- Optimizer: Adam
- Learning Rate: 0.001
- Batch Size: 64
- Number of Epochs: 5

```
model = DigitClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Training and Validation Loop

Assuming `train_loader` and `val_loader` are provided, the training procedure is as follows:

```
num_epochs = 5

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for images, labels in train_loader:
        images = images.view(images.size(0), -1)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    avg_train_loss = running_loss / len(train_loader)
```

```
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in val_loader:
        images = images.view(images.size(0), -1)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

val_accuracy = 100 * correct / total

print(f"Epoch [{epoch+1}/5], "
      f"Train Loss: {avg_train_loss:.4f}, "
      f"Validation Accuracy: {val_accuracy:.2f}%")
```

Why ReLU is Preferred Over Sigmoid and Tanh

ReLU activation is preferred in deep neural networks for the following reasons:

- **Mitigates the vanishing gradient problem:** Unlike Sigmoid and Tanh, ReLU does not saturate for positive values, allowing gradients to propagate effectively in deep networks.
- **Computational efficiency:** ReLU is simple to compute and speeds up training compared to Sigmoid and Tanh activations.

Role of PyTorch Autograd Engine

PyTorch's autograd engine automatically computes gradients for tensors involved in computations. During the forward pass, PyTorch dynamically builds a computation graph that tracks all operations. When `loss.backward()` is called, autograd applies the chain rule to compute gradients for all learnable parameters. These gradients are stored in the `.grad` attribute and are used by the optimizer to update the model parameters.

Conclusion

This implementation demonstrates a complete deep neural network pipeline for handwritten digit classification using PyTorch, including model definition, training, validation, and theoretical understanding of activation functions and automatic differentiation.