Analyzing Factors Influencing Earnings

Salary Prediction for Data Professionals

By- Harshit Ghosh

INTRODUCTION

Importance of Salary Prediction :

- In today's data-driven world, understanding salary trends is crucial for data professionals to make informed career decisions.

- Predicting salaries helps with upskilling, job changes, and salary negotiations.

Data Professional Demand:

- The demand for skilled data professionals is rapidly increasing.
- Salary prediction empowers individuals to navigate their career paths and maximize earning potential.

PROJECT MISSION

01

EXPLORATORY DATA ANALYSIS

UNCOVER PATTERNS & RELATIONSHIPS

02

FEATURE ENGINEERING

CREATE NEW INSIGHTS FROM DATA

03

DATA PREPROCESSING

PREPARE DATA FOR MODELING

04

MACHINE LEARNING MODEL

FIND THE BEST-PERFORMING MODEL

05

MODEL EVALUATION

ASSESS ACCURACY USING METRICS

06

ML PIPELINES

GENERATE PREDICTIONS FOR NEW DATA

EXPLORATORY DATA ANALYSIS (EDA):

01

- Understand data distribution and identify outliers.
- Visualize relationships between variables like experience and salary.
- Summarize key statistics (mean, median, standard deviation).

BASIC HIGHLIGHTS

```
df = pd.read_csv('Salary Prediction of Data Professions.csv')
df.head()
```

Executed at 2024.06.10 22:33:53 in 43ms

5 rows × 13 columns

	FIRST NAME	LAST NAME	SEX	DOJ	CURRENT DATE	DESIGNATION	AGE	SALARY	UNIT	LEAVES USED	LEAVES REMAINING	RATINGS	PAST
0	TOMASA	ARMEN	F	5-18-2014	01-07-2016	Analyst	21.0	44570	Finance	24.0	6.0	2.0	
1	ANNIE	NaN	F	NaN	01-07-2016	Associate	NaN	89207	Web	NaN	13.0	NaN	
2	OLIVE	ANCY	F	7-28-2014	01-07-2016	Analyst	21.0	40955	Finance	23.0	7.0	3.0	
3	CHERRY	AQUILAR	F	04-03-2013	01-07-2016	Analyst	22.0	45550	IT	22.0	8.0	3.0	
4	LEON	ABOULAHOUD	M	11-20-2014	01-07-2016	Analyst	NaN	43161	Operations	27.0	3.0	NaN	

```
df.info()
```

Executed at 2024.06.10 22:33:53 in 17ms

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2639 entries, 0 to 2638
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   FIRST NAME        2639 non-null   object
 1   LAST NAME         2637 non-null   object
 2   SEX               2639 non-null   object
 3   DOJ               2638 non-null   object
 4   CURRENT DATE      2639 non-null   object
 5   DESIGNATION       2639 non-null   object
 6   AGE               2636 non-null   float6
 7   SALARY            2639 non-null   int64
 8   UNIT              2639 non-null   object
 9   LEAVES USED       2636 non-null   float6
 10  LEAVES REMAINING  2637 non-null   float6
 11  RATINGS           2637 non-null   float6
 12  PAST EXP          2639 non-null   int64
dtypes: float64(4), int64(2), object(7)
memory usage: 268.2+ KB
```

```
df.describe()
```

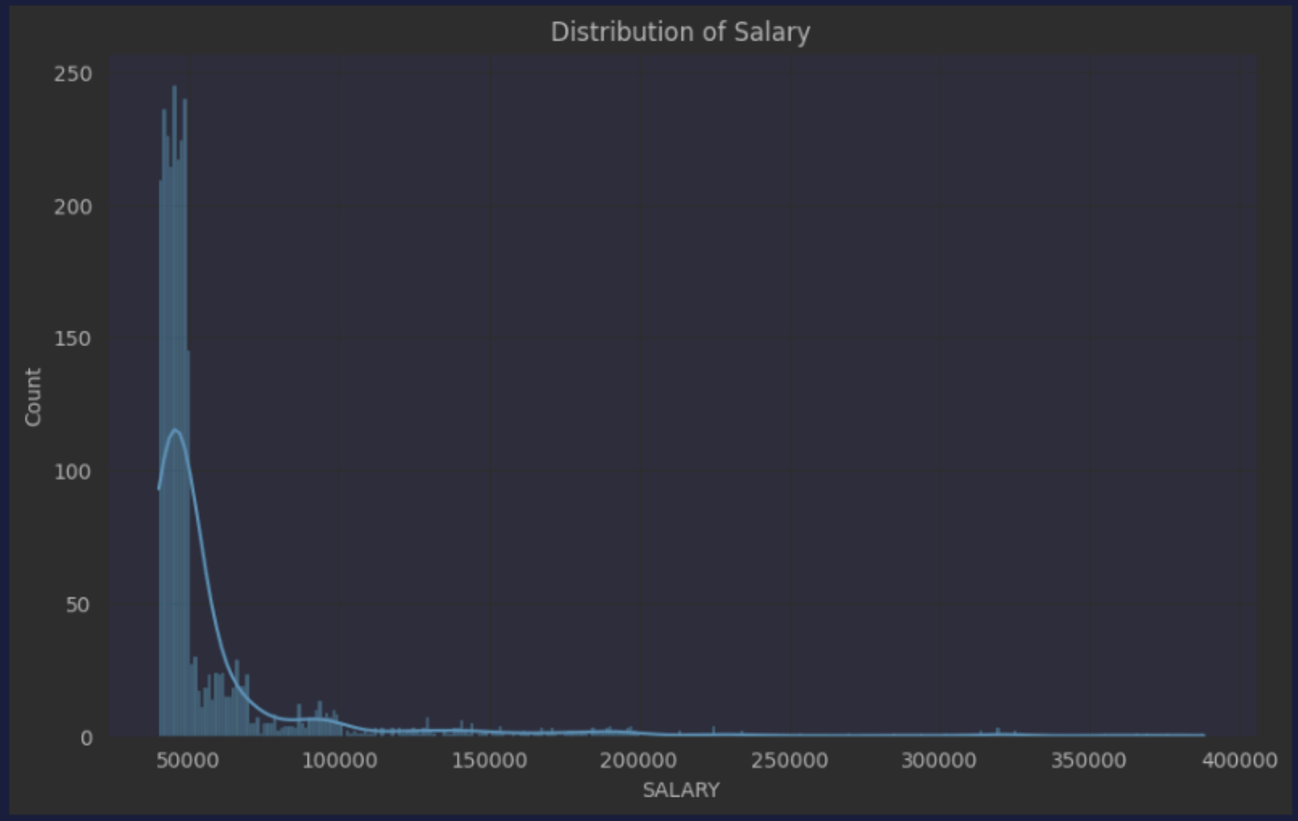
Executed at 2024.06.10 22:57:46 in 14ms

8 rows × 4 columns

	AGE	SALARY	RATINGS	PAST EXP
count	2636.000000	2639.000000	2637.000000	2639.000000
mean	24.756449	58136.678287	3.486159	1.566881
std	3.908228	36876.956944	1.114933	2.728416
min	21.000000	40001.000000	2.000000	0.000000
25%	22.000000	43418.000000	2.000000	0.000000
50%	24.000000	46781.000000	3.000000	1.000000
75%	25.000000	51401.500000	4.000000	2.000000
max	45.000000	388112.000000	5.000000	23.000000

```
# Distribution of Salary
plt.figure(figsize=(10, 6))
sns.histplot(df['SALARY'], kde=True)
plt.title('Distribution of Salary')
plt.show()
```

Executed at 2024.06.10 22:57:47 in 564ms



SALARY DISTRIBUTION

```python
# Scatter plot of Salary vs. Age
plt.figure(figsize=(10, 6))
# Verify the column name 'AGE' exists in your DataFrame. If not, replace it with the correct name
sns.scatterplot(x='AGE', y='SALARY', data=df)  # Pass the DataFrame to the 'data' parameter
plt.title('Salary vs. Age')
plt.show()
```

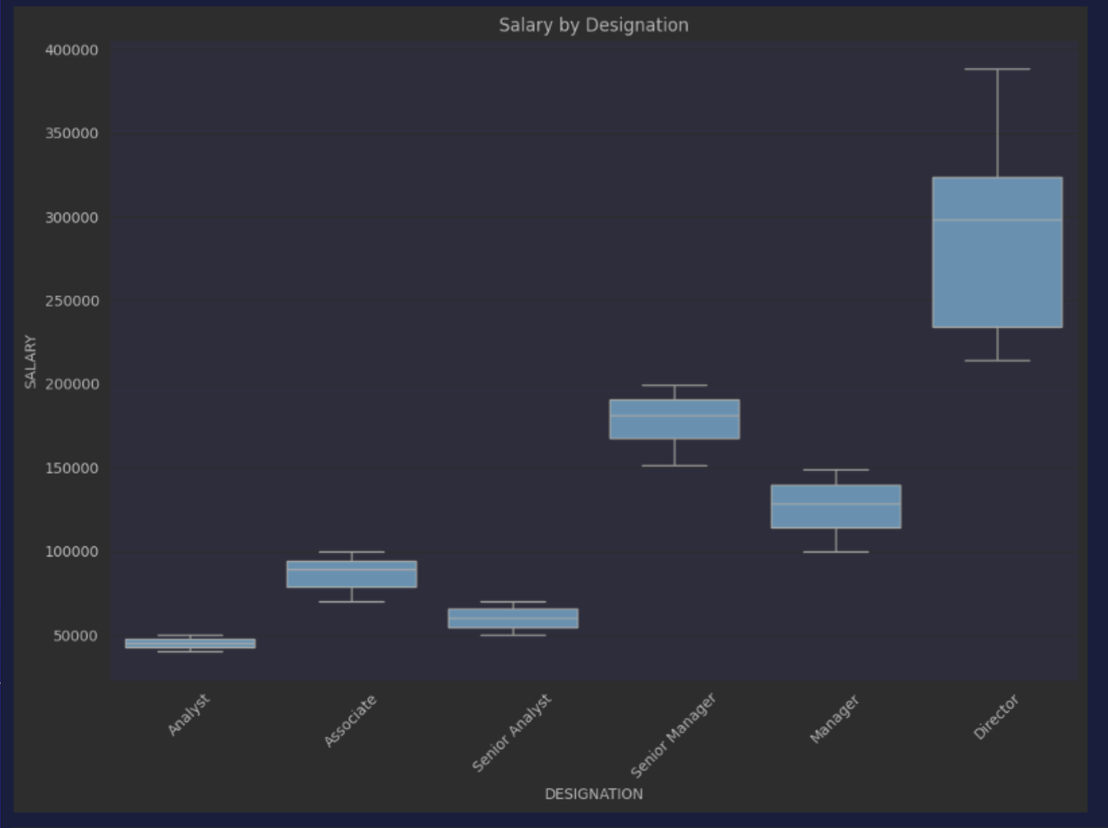
Executed at 2024.06.10 22:57:47 in 243ms

SALARY VS AGE

SALARY AS PER DESIGNATION

```python
# Box plot of Salary by Designation
plt.figure(figsize=(12, 8))
sns.boxplot(x='DESIGNATION', y='SALARY', data=df) # Pass the DataFrame to the 'data' parameter
plt.title('Salary by Designation')
plt.xticks(rotation=45)
plt.show()
```

Executed at 2024.06.10 22:57:47 in 221ms



```
numerical_df = df.select_dtypes(include=['number'])


plt.figure(figsize=(12, 8))
sns.heatmap(numerical_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Executed at 2024.06.10 22:57:46 in 283ms

FEATURE ENGINEERING

01

- Create new features from existing ones (e.g., experience squared).
- Encode categorical variables (e.g., job title to one-hot encoding).
- Transform features to improve model performance (e.g., scaling).

```
# Calculate tenure in years
df['TENURE'] = (df['CURRENT DATE'] - df['DOJ']).dt.days / 365.25
```

- LONGER TENURE GENERALLY CORRELATES WITH HIGHER SALARY,

  INDICATING THATEXPERIENCE WITHIN THE COMPANY IS REWARDED.

```
# Create new features from existing ones (e.g., experience squared).
df['EXP_SQUARED'] = df['PAST EXP'] ** 2
```

```
numeric_features = ['AGE', 'PAST EXP', 'TENURE', 'RATINGS']
numeric_transformer = StandardScaler()
```

Executed at 2024.06.09 11:18:14 in 2ms

- SQUARING EXPERIENCE (EXP^2) IN EDA HELPS CAPTURE POTENTIAL NON-LINEAR TRENDS IN SALARY GROWTH WITH EXPERIENCE, POTENTIALLY IMPROVING MODEL PREDICTIONS.

```
# Convert date columns to datetime
df['DOJ'] = pd.to_datetime(df['DOJ'])
df['CURRENT DATE'] = pd.to_datetime(df['CURRENT DATE'])
```

- COLUMNTRANSFORMER STREAMLINES FEATURE ENGINEERING: COMBINES DIFFERENT PRE-PROCESSING METHODS FOR VARIOUS DATA TYPES (CATEGORICAL & NUMERICAL) INTO A SINGLE STEP.
- UNLOCKS TAILORED TRANSFORMATIONS: ENSURES CATEGORICAL DATA GETS EFFICIENTLY ENCODED (E.G., ONEHOTENCODER) WHILE ALLOWING CUSTOM METHODS FOR NUMERICAL FEATURES.

```
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

Executed at 2024.06.09 11:18:14 in 3ms

```
# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

DATA PREPROCESSING

01

- Prepare the data for model training by :
  - Drop irrelevant columns
  - Handle missing values
  - Encode categorical variables
  - Scale numerical features

```
# Drop irrelevant columns
df.drop(['FIRST NAME', 'LAST NAME', 'SEX', 'LEAVES USED', 'LEAVES REMAINING'], axis=1, inplace=True)
```

Handling Missing Values

```
# Drop rows with missing DOJ values
df.dropna(subset=['DOJ'], inplace=True)

# Replace missing Ratings with the median or a specific value if 0 doesn't make sense
df['RATINGS'].fillna(0, inplace=True)

# Replace missing Age with mean
df['AGE'].fillna(df['AGE'].mean(), inplace=True)
```

```python
# Convert date columns to datetime
df['DOJ'] = pd.to_datetime(df['DOJ'])
df['CURRENT DATE'] = pd.to_datetime(df['CURRENT DATE'])

# Calculate tenure in years
df['TENURE'] = (df['CURRENT DATE'] - df['DOJ']).dt.days / 365.25

# Drop original date columns
df.drop(['DOJ', 'CURRENT DATE'], axis=1, inplace=True)
```

Encode categorical variables

```python
# One-hot encode 'UNIT' and 'DESIGNATION' separately
df = pd.get_dummies(df, columns=['UNIT', 'DESIGNATION'], drop_first=True)
```

Executed at 2024.06.10 22:57:47 in 5ms

```python
df.isnull().sum()
```

Executed at 2024.06.10 22:57:47 in 5ms

1-10 Length: 15, dtype: int64

	<unnamed>
AGE	0
SALARY	0
RATINGS	0
PAST EXP	0
TENURE	0
UNIT_IT	0
UNIT_Management	0
UNIT_Marketing	0
UNIT_Operations	0
UNIT_Web	0

Train-test split data

```
# Define features and target
X = df.drop('SALARY', axis=1)
y = df['SALARY']
```

Executed at 2024.06.10 22:57:47 in 3ms

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

Executed at 2024.06.10 22:57:47 in 5ms

Feature scaling

```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data
X_test_scaled = scaler.transform(X_test)
```

MACHINE LEARNING MODEL DEVELOPMENT

01

- Train various regression models
- Evaluate model performance using metrics like MAE, MSE, RMSE, and R-squared.
- Visualization of predicted vs actual salaries

```
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
}
```

```
predictions = {}
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    predictions[name] = model.predict(X_test_scaled)
```

- EVALUATING THE PERFORMANCE OF EACH SELECTED MODEL

```
def evaluate_model(name, y_test, y_pred):
    print(f"Model: {name}")
    print(f"Mean Absolute Error (MAE): {mean_absolute_error(y_test, y_pred):.2f}")
    print(f"Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred):.2f}")
    print(f"Root Mean Squared Error (RMSE): {mean_squared_error(y_test, y_pred, squared=False):.2f}")
    print(f"R-squared (R2): {r2_score(y_test, y_pred):.2f}")
    print("-" * 30)
```

Executed at 2024.06.10 22:57:48 in 4ms

```
for name, y_pred in predictions.items():
    evaluate_model(name, y_test, y_pred)
```

```
Model: Linear Regression
Mean Absolute Error (MAE): 4296.89
Mean Squared Error (MSE): 63352402.34
Root Mean Squared Error (RMSE): 7959.42
R-squared (R2): 0.96
------------------------------
Model: Decision Tree
Mean Absolute Error (MAE): 4802.52
Mean Squared Error (MSE): 102634324.52
Root Mean Squared Error (RMSE): 10130.86
R-squared (R2): 0.93
------------------------------
Model: Random Forest
Mean Absolute Error (MAE): 4336.54
Mean Squared Error (MSE): 77463484.18
Root Mean Squared Error (RMSE): 8801.33
R-squared (R2): 0.95
------------------------------
Model: Gradient Boosting
Mean Absolute Error (MAE): 4544.06
Mean Squared Error (MSE): 83784569.34
Root Mean Squared Error (RMSE): 9153.39
R-squared (R2): 0.94
------------------------------
```

Model	MAE	MSE	RMSE	R^2
Linear Regression	4296.89	63352402.34	7959.42	0.96
Decision Tree	4811.17	112630152.77	10612.74	0.92
Random Forest	4352.96	81098659.98	9005.48	0.94
Gradient Boosting	4516.27	82563532.72	9086.45	0.94

- FROM THE RESULT WE CHOOSE LINEAR REGRESSION

```
best_model_name = 'Linear Regression'
```

Executed at 2024.06.10 22:57:48 in 2ms

```
best_model = models[best_model_name]
```

Which Model to Choose

- LINEAR REGRESSION: THIS MODEL HAS THE HIGHEST R-SQUARED VALUE (0.96), WHICH INDICATES IT EXPLAINS 96% OF THE VARIANCE IN SALARY. IT ALSO HAS THE LOWEST MEAN ABSOLUTE ERROR (MAE) OF 63,352,402.34, WHICH MEANS THE AVERAGE DIFFERENCE BETWEEN THE PREDICTED SALARY AND THE ACTUAL SALARY IS MINIMIZED.

- DECISION TREE: THIS MODEL HAS A LOWER R-SQUARED VALUE (0.92) COMPARED TO LINEAR REGRESSION, INDICATING IT EXPLAINS LESS OF THE VARIANCE IN SALARY. IT ALSO HAS A HIGHER MAE (48,111.70), SIGNIFYING A HIGHER AVERAGE DIFFERENCE BETWEEN PREDICTED AND ACTUAL SALARIES.

- RANDOM FOREST AND GRADIENT BOOSTING: THESE MODELS HAVE SLIGHTLY LOWER R-SQUARED VALUES (0.94) THAN LINEAR REGRESSION BUT HIGHER MAE VALUES (AROUND 43,500 AND 45,100).

Why Choose This Model

Why Linear Regression might be preferred here:

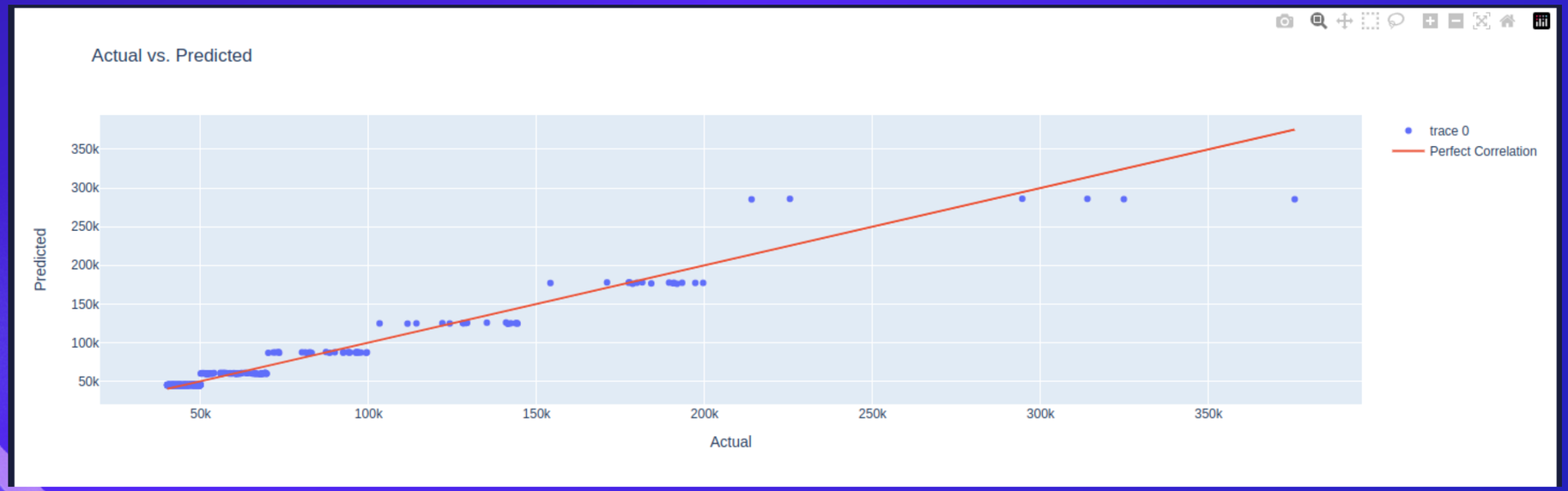
- INTERPRETABILITY: LINEAR REGRESSION IS A SIMPLER MODEL AND EASIER TO INTERPRET. YOU CAN UNDERSTAND HOW EACH FEATURE CONTRIBUTES TO THE PREDICTED SALARY. THIS CAN BE VALUABLE FOR UNDERSTANDING THE FACTORS THAT INFLUENCE SALARY THE MOST.

- ACCURACY: WHILE THE DIFFERENCE IN MAE BETWEEN LINEAR REGRESSION AND OTHER MODELS MIGHT SEEM SIGNIFICANT, IT'S IMPORTANT TO CONSIDER THE SCALE OF THE SALARIES BEING PREDICTED. IN REAL-WORLD SCENARIOS, THESE DIFFERENCES MIGHT BE NEGLIGIBLE.

```python
import plotly.graph_objects as go

predictions = best_model.predict(X_test_scaled)

# Create a scatter plot
fig = go.Figure(data=go.Scatter(x=y_test, y=predictions, mode='markers'))

# Add a line for perfect correlation
fig.add_trace(go.Scatter(x=y_test, y=y_test,
                         mode='lines',
                         name='Perfect Correlation'))

# Update layout
fig.update_layout(title='Actual vs. Predicted',
                  xaxis_title='Actual',
                  yaxis_title='Predicted')

fig.show()
```

Executed at 2024.06.10 23:02:39 in 23ms

- A graph of Actual Value V/S Predicted Value



ML PIPELINES AND MODEL DEPLOYMENT

01

- Create ML pipelines for efficient model training and deployment.
- Deploy the best model to generate predictions

```
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

Executed at 2024.06.09 11:18:14 in 3ms

```
numeric_features = ['AGE', 'PAST EXP', 'TENURE', 'RATINGS']
numeric_transformer = StandardScaler()
```

Executed at 2024.06.09 11:18:14 in 2ms

```
# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

Executed at 2024.06.09 11:18:14 in 2ms

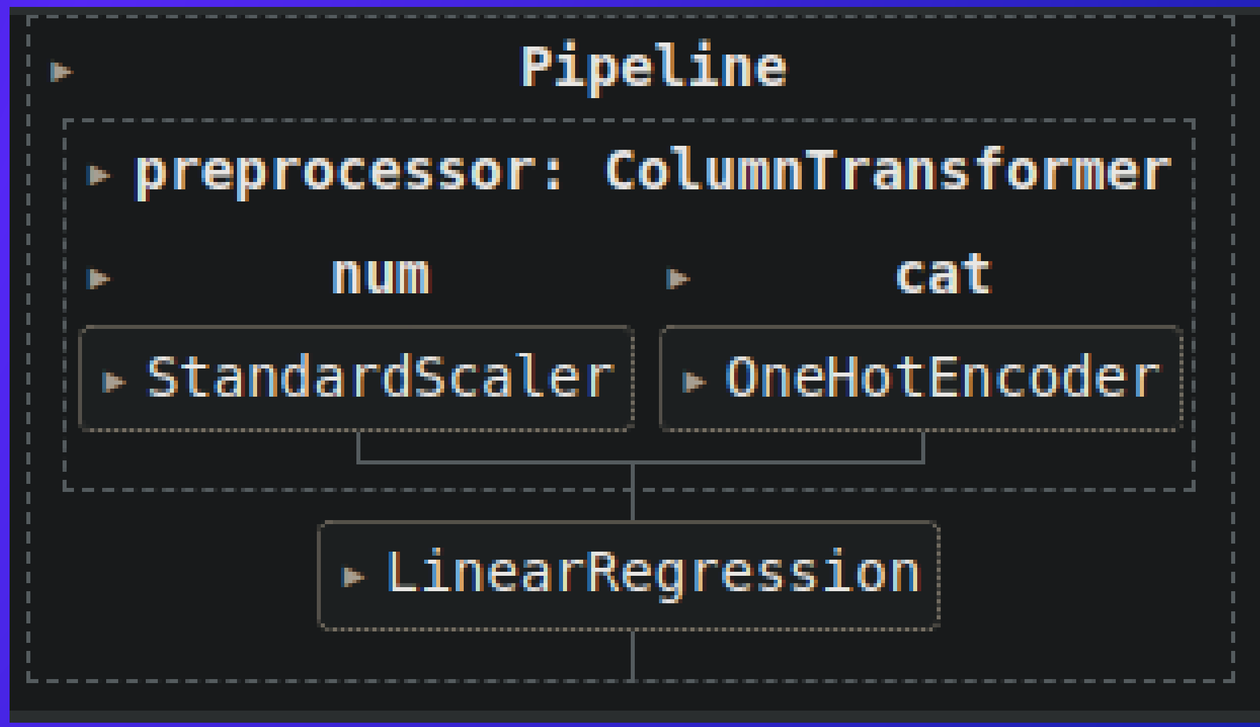
```
# Create a pipeline that first preprocesses the data and then applies the Linear Regression model
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])
```

Executed at 2024.06.09 11:18:14 in 2ms

```
# Split the data again (repeating this step to include the preprocessor in the pipeline)
X = df.drop('SALARY', axis=1)
y = df['SALARY']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Executed at 2024.06.09 11:18:14 in 4ms

```
# Train the pipeline
pipeline.fit(X_train, y_train)
```



```
 #evaluate the pipeline


# Load the pipeline
pipeline = joblib.load('salary_prediction_pipeline_lr.pkl')

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
print("Evaluation metrics for the pipeline:")
print(f"Mean Absolute Error (MAE): {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred):.2f}")
print(f"Root Mean Squared Error (RMSE): {mean_squared_error(y_test, y_pred, squared=False):.2f}")
print(f"R-squared (R2): {r2_score(y_test, y_pred):.2f}")
```

```
Evaluation metrics for the pipeline:
Mean Absolute Error (MAE): 10742.33
Mean Squared Error (MSE): 252845812.19
Root Mean Squared Error (RMSE): 15901.13
R-squared (R2): 0.83
```

RECOMMENDATIONS FOR DATA PROFESSIONALS

01

- Focus on Experience:
  - Gaining experience is crucial, as evidenced by the positive coefficient for 'PAST EXP'.

- Develop In-Demand Skills:
  - Analyze coefficients of features related to specific skills (e.g., 'DESIGNATION' dummies) to identify high-paying skillsets.
  - Consider online courses, bootcamps, certifications, or personal projects to acquire these skills.

RECOMMENDATIONS FOR DATA PROFESSIONALS

02

- Continuous Learning:
  - While formal education ('RATINGS') might have a weaker impact, continuous learning is valuable through workshops, conferences, etc.

- Tenure Considerations:
  - A positive coefficient for 'TENURE' suggests potential benefits of staying with a company for long-term growth.
  - However, be aware of the possibility of career stagnation. Evaluate opportunities for advancement within the company.

THANK YOU!