modeling

October 29, 2024

0.1 Developer Salary Estimator - Modeling

Author: Topaz Montague

Deliverable Description: The goal of this Modeling Deliverable is to create a straightforward yet effective machine learning model to predict developer salaries using core factors like age, education level, years of coding experience, role type, and work experience. We'll start by splitting the data into training and testing sets, focusing on key features, and training models like Linear Regression, Decision Tree, and Random Forest to strike a good balance between accuracy and simplicity. For evaluation, we'll measure each model's performance using metrics such as Mean Absolute Error (MAE) and Mean Squared Error (MSE). The objective is to select a model that not only makes accurate predictions but also sheds light on the main drivers of developer compensation.

Project Artifacts: GitHub Repository Link: Developer Salary GitHub Repository

Overleaf Project Report: Developer Salary Overleaf Project Report

Shiny App Dashboard: Developer Salary Shiny App Dashboard - PLACEHOLDER

Data Preparation for Modeling

Load the Necessary Libraries and Data

[201]:
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Load the dataset
data_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(data_path)

# Display the first few rows to confirm it's loaded correctly
print(data.head())
```

```
   ResponseId  Age Range RemoteWork       EdLevel  YearsCode  \
0         390         30     Remote  Some college          7
```

```
1        399        50     Remote  Some college        38
2        417        40     Remote       Masters        21
3        427        20     Remote     Bachelors         9
4        429        30     Remote     Bachelors        20

                                        DevType OrgSize  \
0                                       Student      15
1                          Developer, full-stack    2500
2                            Developer, back-end     250
3  Developer, embedded applications or devices    2500
4                      Engineer, site reliability     250

                     Country                 ICorPM  WorkExp  …  \
0  United States of America  Individual contributor        8  …
1  United States of America  Individual contributor       30  …
2                     Brazil  Individual contributor       17  …
3                    Ukraine  Individual contributor        4  …
4  United States of America  Individual contributor       15  …

  Database_Oracle  Database_PostgreSQL Database_Presto Database_RavenDB  \
0              No                   No              No               No
1              No                  Yes              No               No
2              No                  Yes              No               No
3              No                   No              No               No
4              No                   No              No               No

  Database_Redis Database_SQLite Database_Snowflake Database_Solr  \
0             No             Yes                 No            No
1             No              No                Yes            No
2             No              No                 No            No
3             No              No                 No            No
4            Yes             Yes                 No            No

  Database_Supabase Database_TiDB
0                No            No
1                No            No
2                No            No
3                No            No
4                No            No

[5 rows x 96 columns]
```

Prepare the Data for Modeling

[202]:
```python
# Step 1: Select Relevant Columns
selected_columns = ['TotalComp', 'Age Range', 'EdLevel', 'YearsCode',␣
 ↪'DevType', 'OrgSize', 'WorkExp', 'Industry']
data = data[selected_columns]
```

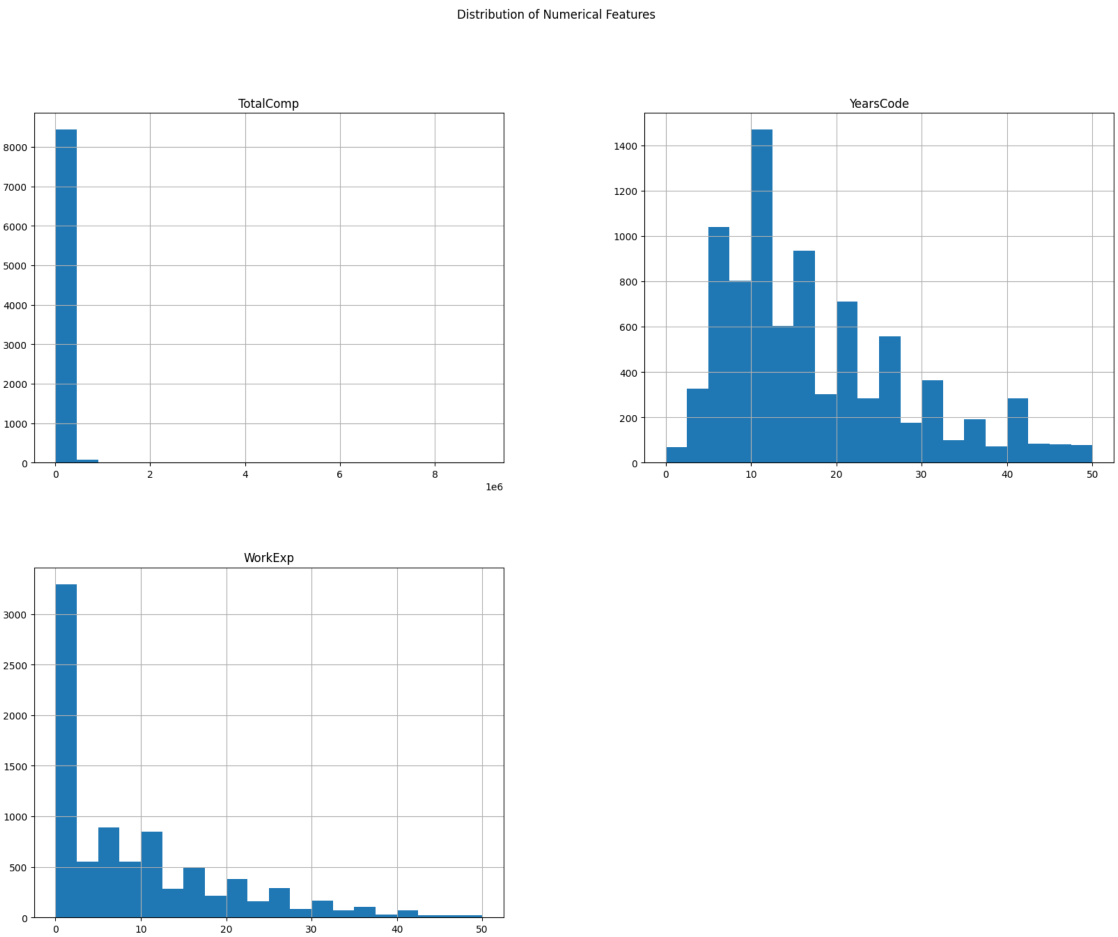
```python
# Step 2: Handle Missing Values

# Drop rows where TotalComp is missing (since it's the target variable)
data = data.dropna(subset=['TotalComp'])

# Handle missing values in other columns
# Check proportion of missing values
missing_values = data.isnull().mean()

# Drop or impute missing values based on proportion threshold
threshold = 0.1  # example threshold: if more than 10% of values are missing,␣
↪drop the column; otherwise, drop rows
columns_to_drop = missing_values[missing_values > threshold].index
data = data.drop(columns=columns_to_drop)

# For columns with fewer missing values, drop rows with missing values
data = data.dropna()

# Step 3: Encode Categorical Variables

# List of categorical columns
categorical_columns = ['Age Range', 'EdLevel', 'DevType', 'Industry']

# Apply one-hot encoding for linear models or label encoding for tree-based␣
↪models
use_one_hot = True  # Set this to False if using tree-based models

if use_one_hot:
    # Use OneHotEncoder for linear models
    data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)
else:
    # Use LabelEncoder for tree-based models
    label_encoders = {}
    for column in categorical_columns:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le  # Store encoder for potential inverse␣
↪transformation

# Display the prepared data
print(data.head())
```

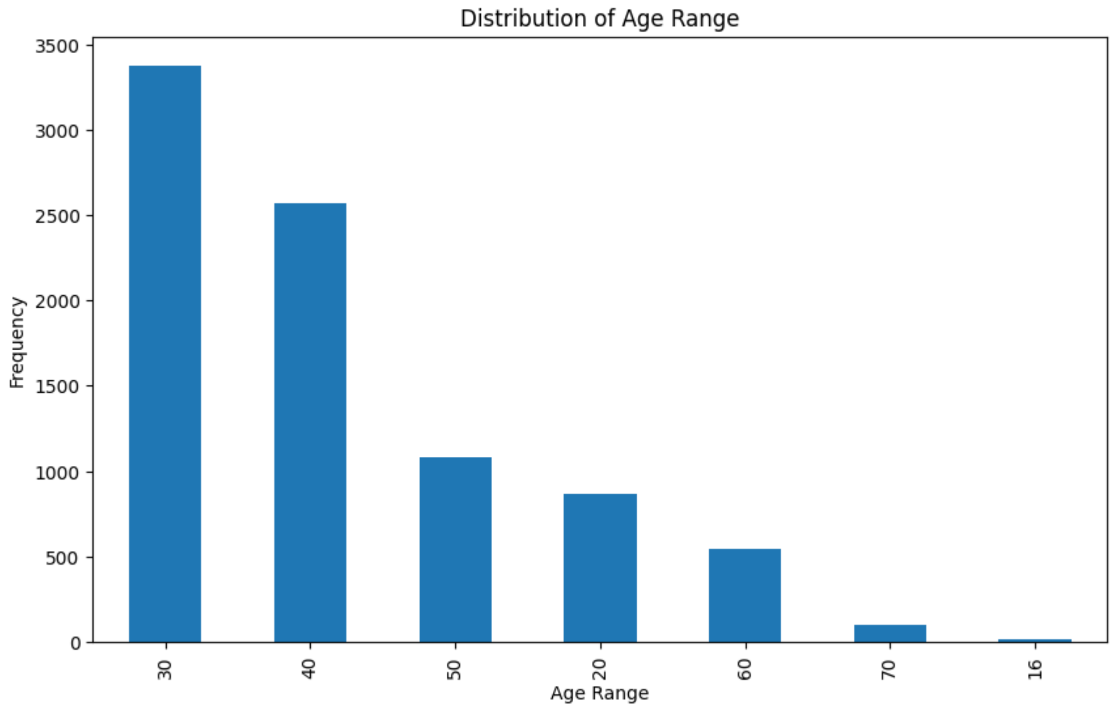
```
   TotalComp  YearsCode OrgSize  WorkExp  Age Range_20  Age Range_30  \
0     110000          7      15        8         False          True
1     195000         38    2500       30         False         False
2     170000         21     250       17         False         False
```

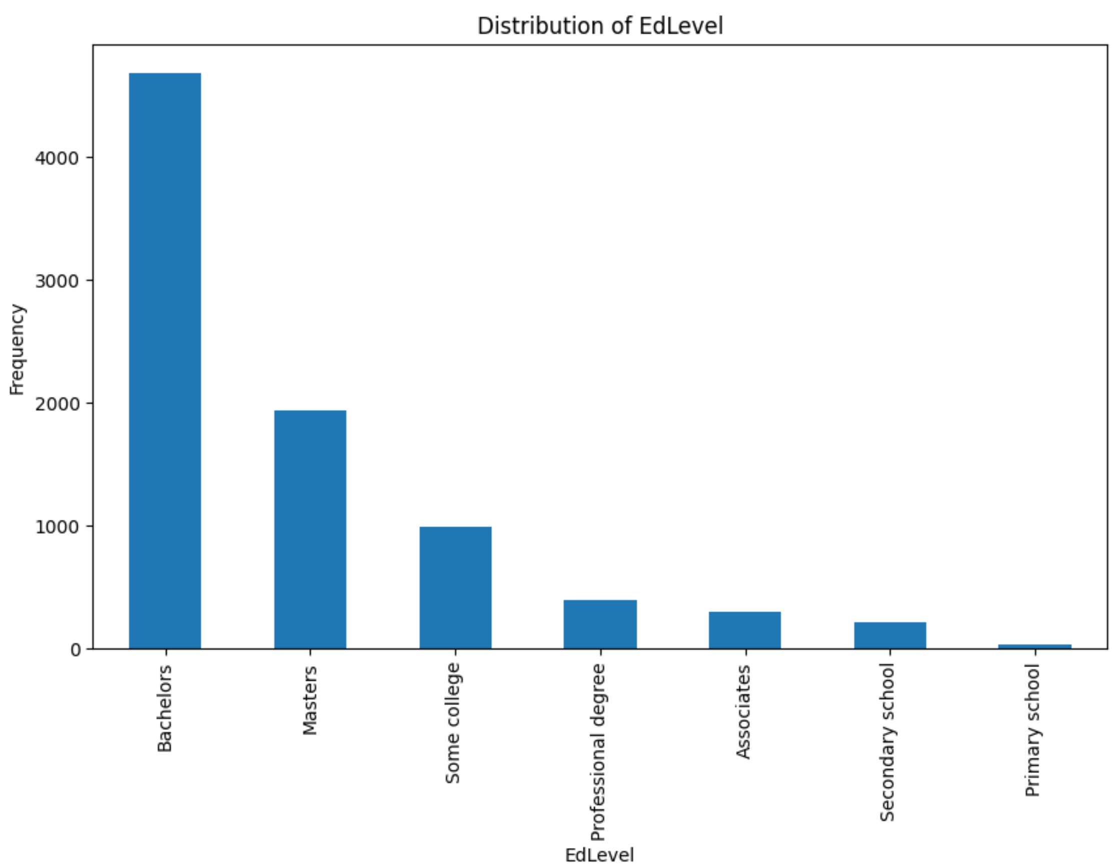
```
3     50000          9    2500        4         True         False
4    230000         20     250       15        False          True

   Age Range_40  Age Range_50  Age Range_60  Age Range_70  …  \
0         False         False         False         False  …
1         False          True         False         False  …
2          True         False         False         False  …
3         False         False         False         False  …
4         False         False         False         False  …

   Industry_Healthcare  Industry_Higher Education  Industry_Insurance  \
0                False                      False               False
1                 True                      False               False
2                False                      False               False
3                False                      False               False
4                False                      False               False

   Industry_Internet Telecomm or Information Services  Industry_Manufacturing  \
0                                              False                   False
1                                              False                   False
2                                              False                   False
3                                              False                   False
4                                              False                   False

   Industry_Media & Advertising Services  Industry_Other  \
0                                 False           False
1                                 False           False
2                                 False            True
3                                 False           False
4                                 False           False

   Industry_Retail and Consumer Services  Industry_Software Development  \
0                                 False                          False
1                                 False                          False
2                                 False                          False
3                                 False                           True
4                                 False                           True

   Industry_Transportation or Supply Chain
0                                     True
1                                    False
2                                    False
3                                    False
4                                    False

[5 rows x 63 columns]
```

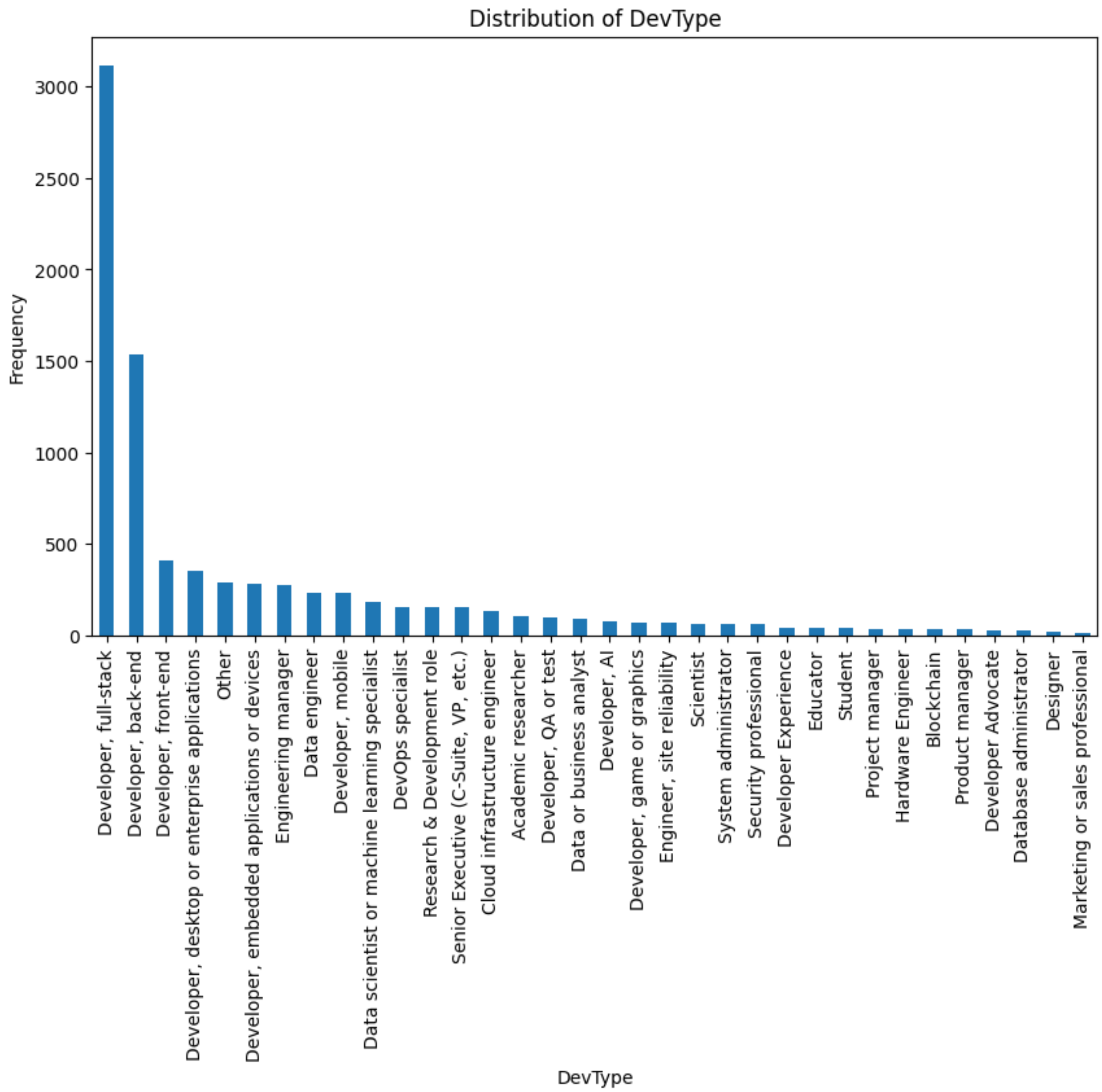
Get Basic Information on the Featurers Use hist() to look at the distribution of the selected features

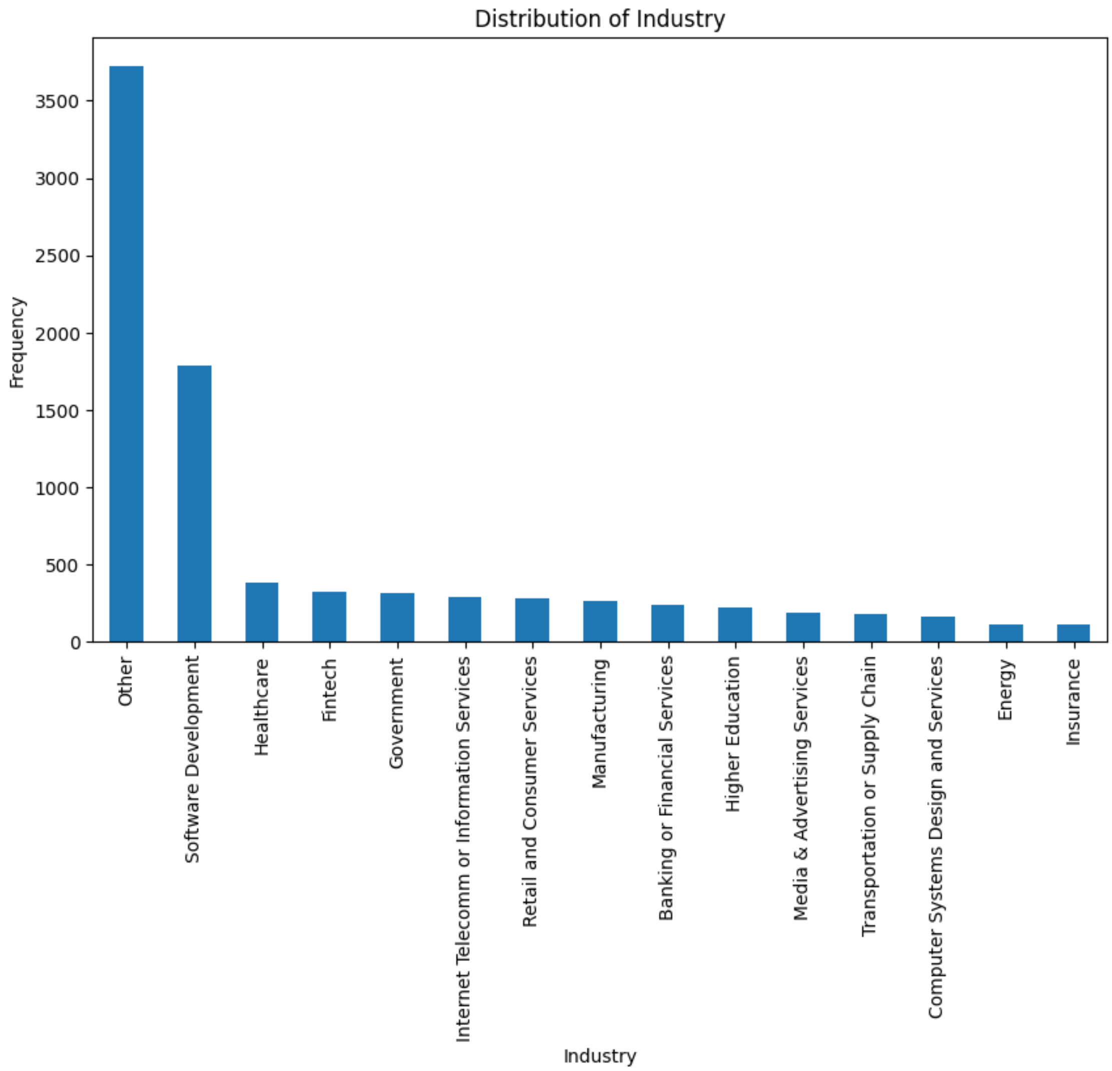
[203]:
```python
# Load the dataset
df = pd.read_csv('data/cleaned/Transformed_Developer_Survey_Data.csv')

# Select only the relevant columns
selected_columns = ["TotalComp", "Age Range", "EdLevel", "YearsCode",␣
↪"DevType", "OrgSize", "WorkExp", "Industry"]
df = df[selected_columns]

# Separate numerical and categorical columns
numerical_columns = ["TotalComp", "YearsCode", "OrgSize", "WorkExp"]
categorical_columns = ["Age Range", "EdLevel", "DevType", "Industry"]

# Plot histograms for numerical features
df[numerical_columns].hist(bins=20, figsize=(20, 15))
plt.suptitle("Distribution of Numerical Features")
plt.show()

# Plot bar charts for categorical features
for col in categorical_columns:
    plt.figure(figsize=(10, 6))
    df[col].value_counts().plot(kind='bar')
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```


Distribution of Numerical Features
TotalComp
0 1000 2000 3000 4000 5000 6000 7000 8000
0 2 4 6 8
1e6
YearsCode
0 200 400 600 800 1000 1200 1400
0 10 20 30 40 50
WorkExp
0 500 1000 1500 2000 2500 3000
0 10 20 30 40 50


Distribution of Age Range
Frequency
3500
3000
2500
2000
1500
1000
500
0
30
40
50
20
60
70
16
Age Range


Distribution of EdLevel
Frequency
4000
3000
2000
1000
0
Bachelors
Masters
Some college
Professional degree
Associates
Secondary school
Primary school
EdLevel


Distribution of DevType
Frequency
0
500
1000
1500
2000
2500
3000
Developer, full-stack
Developer, back-end
Developer, front-end
Developer, desktop or enterprise applications
Other
Developer, embedded applications or devices
Engineering manager
Data engineer
Developer, mobile
Data scientist or machine learning specialist
DevOps specialist
Research & Development role
Senior Executive (C-Suite, VP, etc.)
Cloud infrastructure engineer
Academic researcher
Developer, QA or test
Data or business analyst
Developer, AI
Developer, game or graphics
Engineer, site reliability
Scientist
System administrator
Security professional
Developer Experience
Educator
Student
Project manager
Hardware Engineer
Blockchain
Product manager
Developer Advocate
Database administrator
Designer
Marketing or sales professional
DevType



[204]:
```
# Check that cleaning is good
df.head(n=10)
```

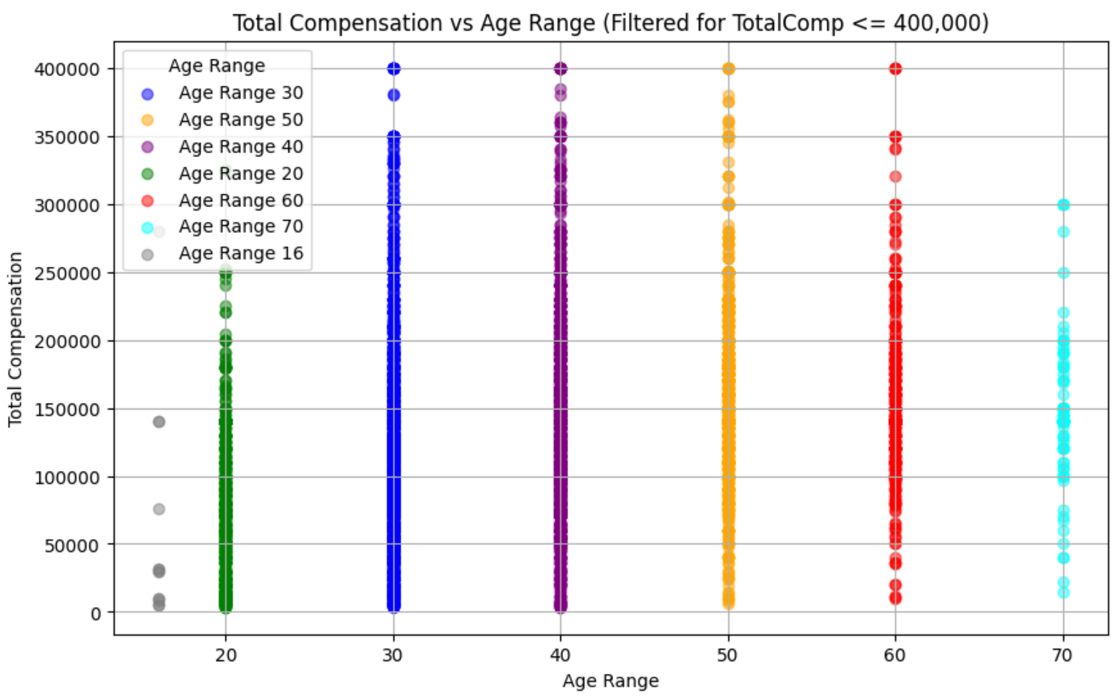
[204]:
```
   TotalComp  Age Range          EdLevel  YearsCode  \
0     110000         30     Some college          7
1     195000         50     Some college         38
2     170000         40          Masters         21
3      50000         20        Bachelors          9
4     230000         30        Bachelors         20
5      85000         40        Bachelors         25
6     160000         40  Secondary school         20
7     110000         60        Bachelors         25
8     190000         40          Masters         23
9     115000         50       Associates         10
```

```
                                     DevType OrgSize  WorkExp  \
0                                    Student      15        8
1                       Developer, full-stack    2500       30
2                         Developer, back-end     250       17
3  Developer, embedded applications or devices    2500        4
4                  Engineer, site reliability     250       15
5                       Developer, full-stack      50       25
6                       Developer, full-stack      15       20
7                        Developer, front-end    2500        0
8                         Engineering manager   10000        0
9                       Developer, full-stack     250       10

                        Industry
0  Transportation or Supply Chain
1                      Healthcare
2                           Other
3            Software Development
4            Software Development
5    Retail and Consumer Services
6            Software Development
7                           Other
8                           Other
9                           Other
```

Plot to Visualize the Selected Features vs Total Compensation Total Compensation vs Age Range

[205]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(file_path)

# Filter data to include only rows where TotalComp <= 400,000
data = data[data['TotalComp'] <= 400000]

# Define color mapping (specific to the column you're working with)
color_map = {
    20: 'green', 30: 'blue', 40: 'purple', 50: 'orange', 60: 'red', 70: 'cyan'
}

# Plotting example for TotalComp vs Age Range
plt.figure(figsize=(10, 6))

# Loop through unique age ranges to create separate scatter plots
```

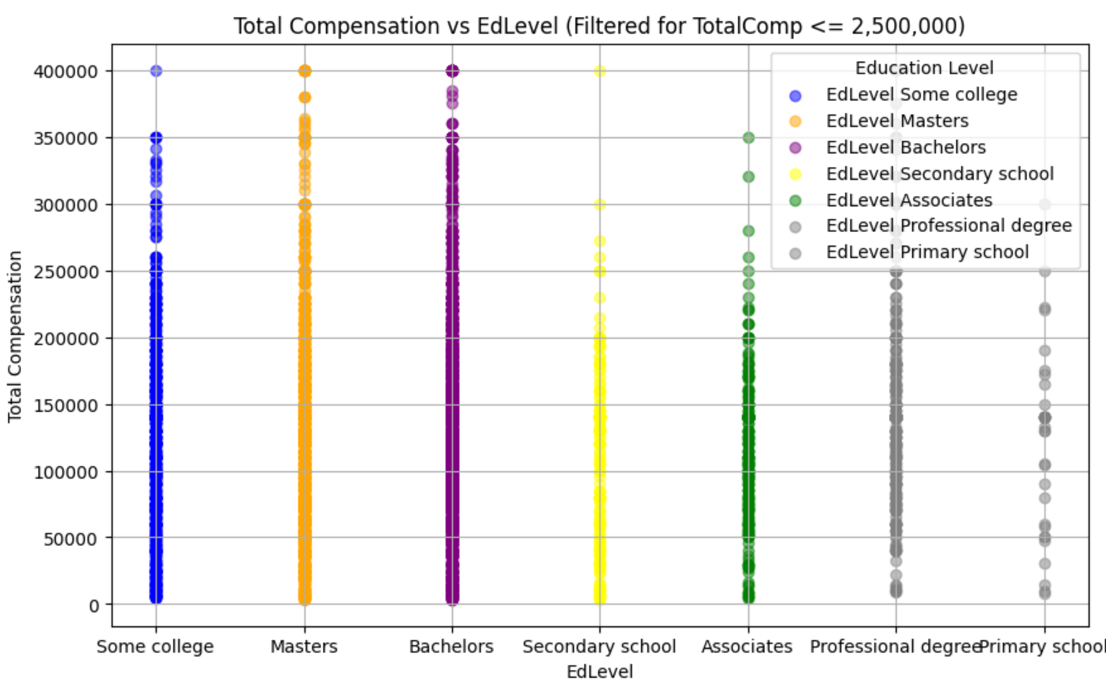
```
for age_range in data['Age Range'].unique():
    subset = data[data['Age Range'] == age_range]
    plt.scatter(
        subset['Age Range'],
        subset['TotalComp'],
        c=color_map.get(age_range, 'grey'),  # Default to 'grey' if age_range␣
↪not in color_map
        alpha=0.5,
        label=f'Age Range {age_range}'
    )

plt.title('Total Compensation vs Age Range (Filtered for TotalComp <= 400,000)')
plt.xlabel('Age Range')
plt.ylabel('Total Compensation')

# Set y-axis to display values in plain format (no scientific notation)
plt.ticklabel_format(style='plain', axis='y')

plt.legend(title='Age Range')
plt.grid(True)
plt.show()
```



Total Compensation vs Education Level

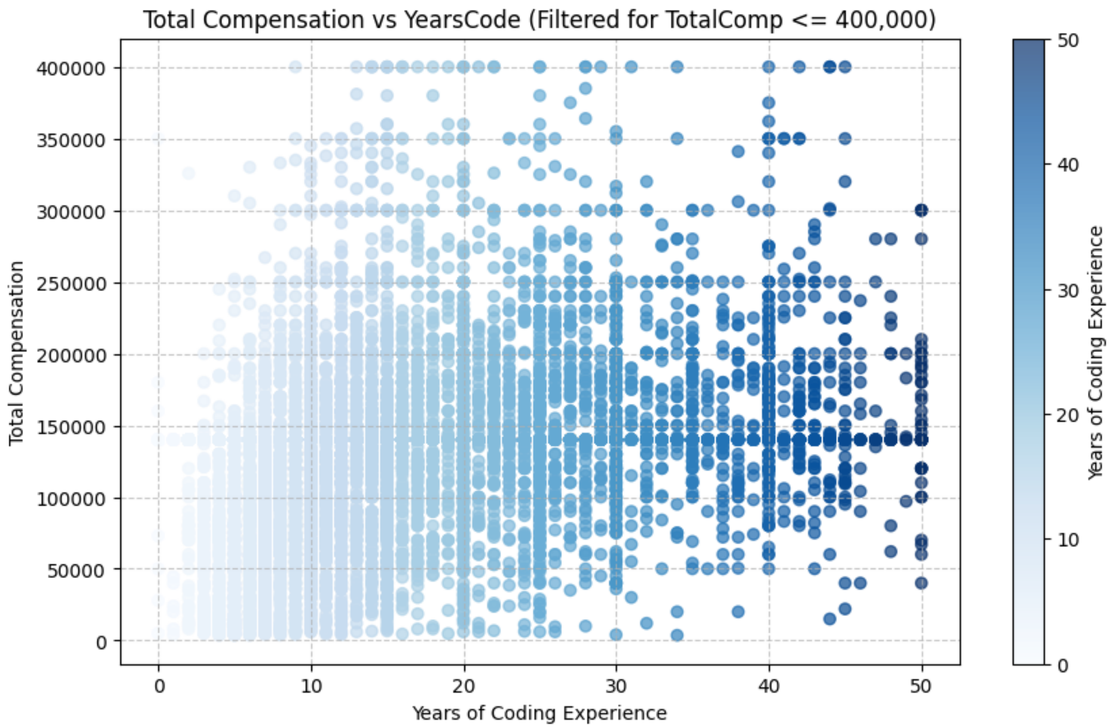
[206]:

```python
# Define color mapping for EdLevel
color_map = {
    'High school': 'green', 'Some college': 'blue', 'Bachelors': 'purple',
    'Masters': 'orange', 'Doctorate': 'red', 'Secondary school': 'yellow',␣
 ↪'Associates': 'green', 'Professional Degree': 'purple'
}

plt.figure(figsize=(10, 6))
for ed_level in data['EdLevel'].unique():
    subset = data[data['EdLevel'] == ed_level]
    plt.scatter(
        subset['EdLevel'],
        subset['TotalComp'],
        c=color_map.get(ed_level, 'grey'),
        alpha=0.5,
        label=f'EdLevel {ed_level}'
    )

plt.title('Total Compensation vs EdLevel (Filtered for TotalComp <= 2,500,000)')
plt.xlabel('EdLevel')
plt.ylabel('Total Compensation')
plt.ticklabel_format(style='plain', axis='y')
plt.legend(title='Education Level')
plt.grid(True)
plt.show()
```



Total Compensation vs Years of Coding Experience

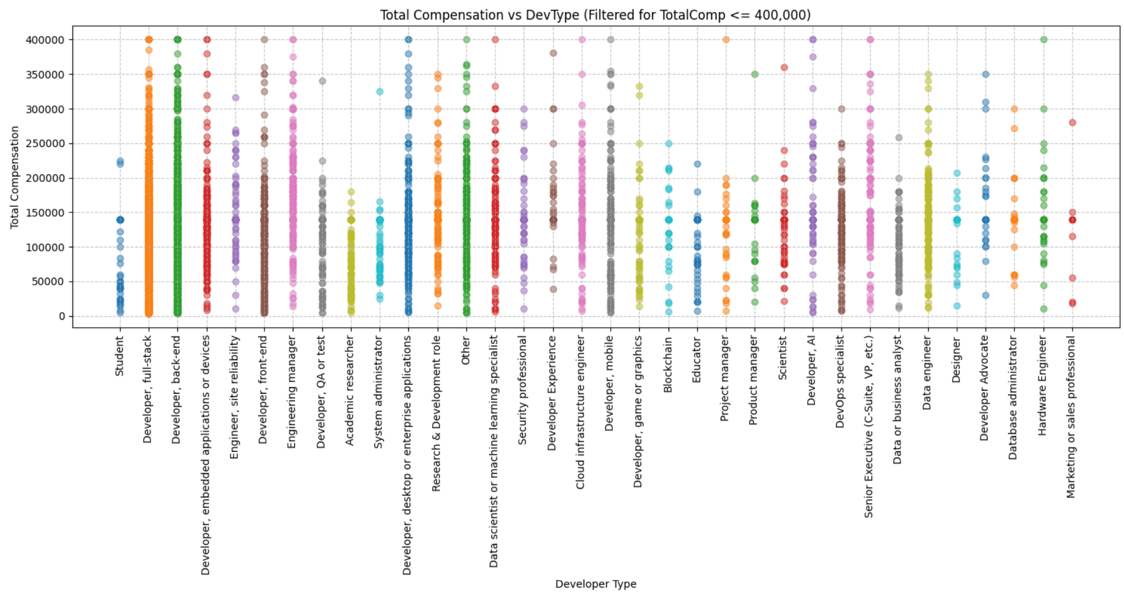
[207]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# Load the dataset
file_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(file_path)

# Filter dataset to only include TotalComp <= 400,000
filtered_data = data[data['TotalComp'] <= 400000]

# Define figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

# Normalize the YearsCode data for the color map
norm = plt.Normalize(filtered_data['YearsCode'].min(),␣
 ↪filtered_data['YearsCode'].max())
colors = cm.Blues(norm(filtered_data['YearsCode']))

# Plotting TotalComp vs YearsCode with a gradient blue color based on YearsCode
scatter = ax.scatter(
    filtered_data['YearsCode'],
    filtered_data['TotalComp'],
    c=filtered_data['YearsCode'],
    cmap='Blues',
    alpha=0.7
)

# Add color bar to indicate the gradient scale of YearsCode
cbar = fig.colorbar(scatter, ax=ax)
cbar.set_label('Years of Coding Experience')

# Set titles and labels
ax.set_title('Total Compensation vs YearsCode (Filtered for TotalComp <=␣
 ↪400,000)')
ax.set_xlabel('Years of Coding Experience')
ax.set_ylabel('Total Compensation')

# Display y-axis in plain format without scientific notation
ax.ticklabel_format(style='plain', axis='y')

# Add grid for readability
ax.grid(True, linestyle='--', alpha=0.7)

plt.show()
```



Total Compensation vs Type of Developer

[208]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(file_path)

# Filter dataset to only include TotalComp <= 400,000
filtered_data = data[data['TotalComp'] <= 400000]

# Define figure with adjusted width
plt.figure(figsize=(15, 8))

# Plotting TotalComp vs DevType with appropriate adjustments for readability
for dev_type in filtered_data['DevType'].unique():
    subset = filtered_data[filtered_data['DevType'] == dev_type]
    plt.scatter(
        subset['DevType'],
        subset['TotalComp'],
        alpha=0.5,
        label=f'DevType {dev_type}'
    )
```

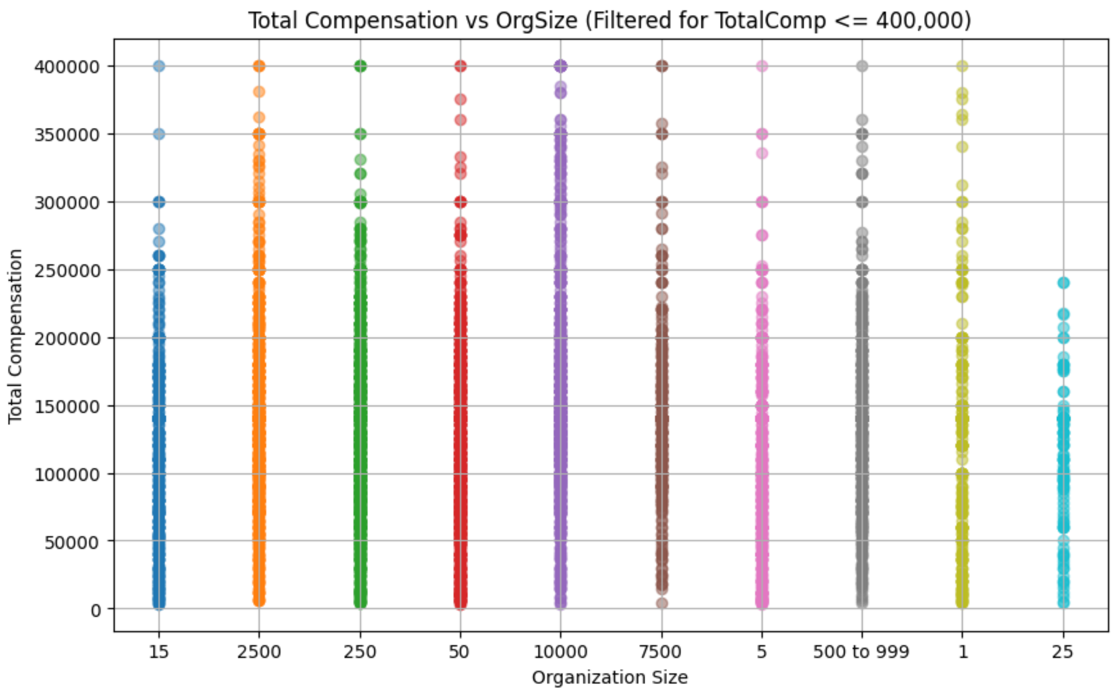
```python
# Title and labels
plt.title('Total Compensation vs DevType (Filtered for TotalComp <= 400,000)')
plt.xlabel('Developer Type')
plt.ylabel('Total Compensation')

# Display y-axis in plain format without scientific notation
plt.ticklabel_format(style='plain', axis='y')

# Rotate x-axis labels for readability
plt.xticks(rotation=90)

# Add grid for readability
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()  # Adjust layout to fit everything neatly
plt.show()
```



Total Compensation vs Organization Size

[209]:
```python
plt.figure(figsize=(10, 6))
for org_size in filtered_data['OrgSize'].unique():
    subset = filtered_data[filtered_data['OrgSize'] == org_size]
    plt.scatter(
        subset['OrgSize'],
        subset['TotalComp'],
        alpha=0.5,
        label=f'OrgSize {org_size}'
```

```
    )

plt.title('Total Compensation vs OrgSize (Filtered for TotalComp <= 400,000)')
plt.xlabel('Organization Size')
plt.ylabel('Total Compensation')
plt.ticklabel_format(style='plain', axis='y')
plt.grid(True)
plt.show()
```



Declare the Train-Test Split

[210]:
```
# Use train_test_split from sklearn.model_selection to split the data into␣
↪training and testing sets. A typical split is 80% for training and 20% for␣
↪testing:
from sklearn.model_selection import train_test_split

# Filter data to include only rows where TotalComp <= 400000
filtered_data = data[data['TotalComp'] <= 400000]

# Define features and target
X = filtered_data[['Age Range', 'EdLevel', 'YearsCode', 'DevType', 'OrgSize',␣
↪'WorkExp', 'Industry']]
y = filtered_data['TotalComp']

# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

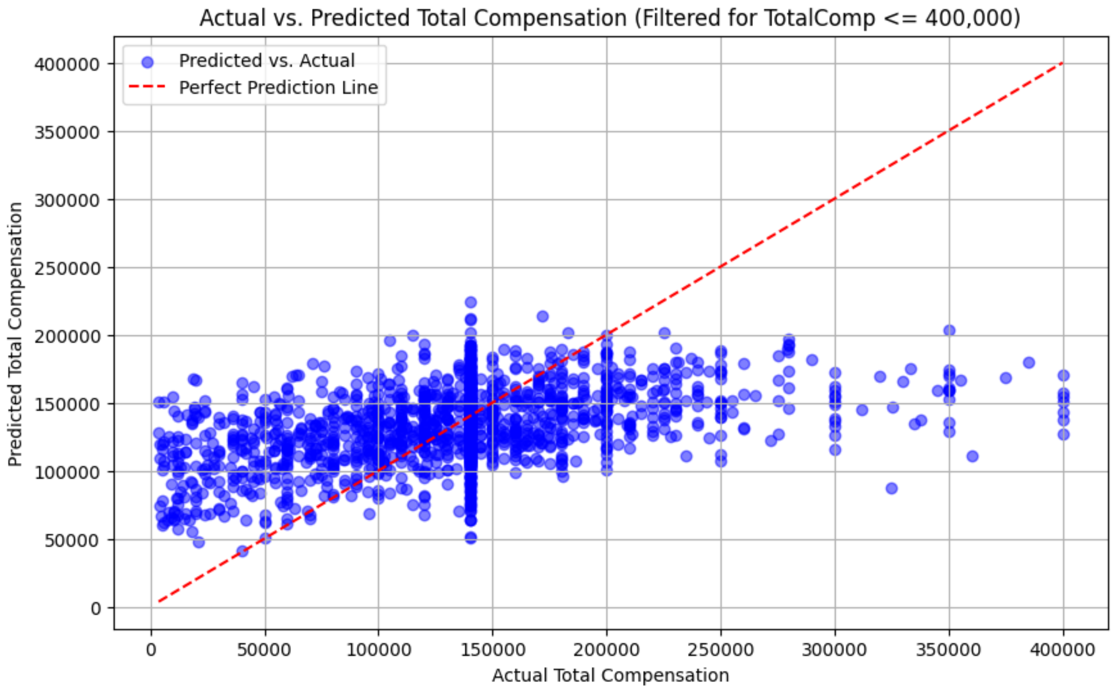
Model Training and Evaluation Linear Regression

[211]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv('data/cleaned/Transformed_Developer_Survey_Data.csv')

# Filter data to include only rows where TotalComp <= 400000
data = data[data['TotalComp'] <= 400000]

# Select relevant columns
selected_columns = ['TotalComp', 'Age Range', 'EdLevel', 'YearsCode',␣
↪'DevType', 'OrgSize', 'WorkExp', 'Industry']
data = data[selected_columns]

# Handle missing values (drop rows with missing values for simplicity)
data = data.dropna()

# One-hot encode categorical variables (including 'OrgSize')
data = pd.get_dummies(data, columns=['Age Range', 'EdLevel', 'DevType',␣
↪'OrgSize', 'Industry'], drop_first=True)

# Define features and target
X = data.drop('TotalComp', axis=1)
y = data['TotalComp']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
MAE: 41487.11481678698
MSE: 3243122526.822035
R2 Score: 0.19654005605061176
```

Linear Regression Model Graph for Total Compensation Predictions vs Actuals

[212]:
```python
import matplotlib.pyplot as plt

# Filter the data for TotalComp <= 400000
y_test_filtered = y_test[y_test <= 400000]
y_pred_filtered = y_pred[y_test <= 400000]  # Apply the same mask to y_pred

# Scatter plot of actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test_filtered, y_pred_filtered, alpha=0.5, color='blue',␣
 ↪label='Predicted vs. Actual')
plt.plot([y_test_filtered.min(), y_test_filtered.max()],
         [y_test_filtered.min(), y_test_filtered.max()],
         color='red', linestyle='--', label='Perfect Prediction Line')

# Labeling the plot
plt.xlabel("Actual Total Compensation")
plt.ylabel("Predicted Total Compensation")
plt.title("Actual vs. Predicted Total Compensation (Filtered for TotalComp <=␣
 ↪400,000)")
plt.legend()
plt.grid(True)
plt.ticklabel_format(style='plain', axis='y')  # Avoid scientific notation on␣
 ↪the y-axis
plt.show()
```



Decision Tree Regressor

[213]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
file_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(file_path)

# Filter data to include only rows where TotalComp <= 400000
data = data[data['TotalComp'] <= 400000]

# Select relevant columns
selected_columns = ['TotalComp', 'Age Range', 'EdLevel', 'YearsCode',␣
↪'DevType', 'OrgSize', 'WorkExp', 'Industry']
data = data[selected_columns]

# Handle missing values
data = data.dropna(subset=['TotalComp'])  # Drop rows with missing TotalComp␣
↪values
data = data.dropna()  # Drop rows with missing values in other columns

# Encode categorical variables
```

```
categorical_columns = ['Age Range', 'EdLevel', 'DevType', 'OrgSize', 'Industry']
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# Define features and target
X = data.drop('TotalComp', axis=1)
y = data['TotalComp']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize and train the Decision Tree Regressor model
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

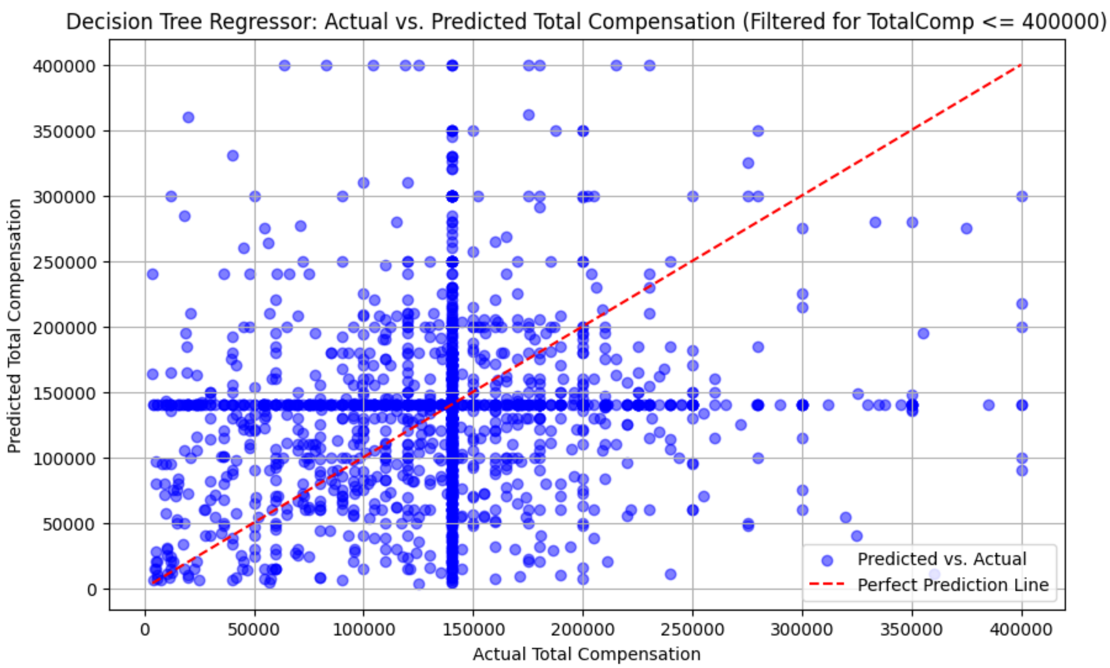
```
MAE: 60327.253656967536
MSE: 6833384163.111126
R2 Score: -0.6929210695156642
```

Decision Tree Regressor Model Graph for Total Compensation Predictions vs Actuals

[214]:

```
# Visualize Actual vs Predicted TotalComp for values <= 400000
import matplotlib.pyplot as plt

# Filter the data for TotalComp <= 400000
y_test_filtered = y_test[y_test <= 400000]
y_pred_filtered = y_pred[y_test <= 400000]  # Apply the same mask to y_pred

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test_filtered, y_pred_filtered, alpha=0.5, color='blue',␣
↪label='Predicted vs. Actual')
plt.plot([y_test_filtered.min(), y_test_filtered.max()], [y_test_filtered.
↪min(), y_test_filtered.max()],
         color='red', linestyle='--', label='Perfect Prediction Line')

# Labeling the plot
plt.xlabel("Actual Total Compensation")
plt.ylabel("Predicted Total Compensation")
```

```
plt.title("Decision Tree Regressor: Actual vs. Predicted Total Compensation␣
↪(Filtered for TotalComp <= 400000)")
plt.legend()
plt.grid(True)
plt.ticklabel_format(style='plain')
plt.show()
```



Random Forest Regressor

[215]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
file_path = 'data/cleaned/Transformed_Developer_Survey_Data.csv'
data = pd.read_csv(file_path)

# Filter data to include only rows where TotalComp <= 400000
data = data[data['TotalComp'] <= 400000]

# Select relevant columns
selected_columns = ['TotalComp', 'Age Range', 'EdLevel', 'YearsCode',␣
↪'DevType', 'OrgSize', 'WorkExp', 'Industry']
```

```
data = data[selected_columns]

# Handle missing values (drop rows with missing values for simplicity)
data = data.dropna()

# Encode categorical variables
categorical_columns = ['Age Range', 'EdLevel', 'DevType', 'OrgSize', 'Industry']
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# Define features and target
X = data.drop('TotalComp', axis=1)
y = data['TotalComp']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize and train the Random Forest Regressor model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
MAE: 45342.22948174411
MSE: 3782466515.6126447
R2 Score: 0.06292151792285938
```

Random Forest Regressor Model Graph for Total Compensation Predictions vs Actuals

[216]:
```
import matplotlib.pyplot as plt

# Filter to only include TotalComp values <= 400,000
y_test_filtered = y_test[y_test <= 400000]
y_pred_filtered = y_pred[y_test <= 400000]  # Apply the same mask to y_pred

# Visualize Actual vs Predicted TotalComp (Filtered for <= 400,000)
plt.figure(figsize=(10, 6))

# Scatter plot of actual vs predicted values
plt.scatter(y_test_filtered, y_pred_filtered, alpha=0.5, color='blue',␣
↪label='Predicted vs. Actual')
```

```python
# Plot a perfect prediction line
plt.plot([y_test_filtered.min(), y_test_filtered.max()], [y_test_filtered.
 ↪min(), y_test_filtered.max()],
         color='red', linestyle='--', label='Perfect Prediction Line')

# Labeling the plot
plt.xlabel("Actual Total Compensation")
plt.ylabel("Predicted Total Compensation")
plt.title("Random Forest Regressor: Actual vs. Predicted Total Compensation␣
 ↪(Filtered for TotalComp <= 400,000)")
plt.legend()
plt.grid(True)

# Set y-axis to display values in plain format (no scientific notation)
plt.ticklabel_format(style='plain', axis='both')
plt.show()
```