configuration, and designing education and training interventions for skills mismatch. For example, prior work using the OEWS data have investigated aspects of wage discontinuity and the relationship between wages and industry growth. This dataset does exactly that by giving a first ever glimpse into the complex reality of jobs and wages. It's a street plan that not only reveals where individuals are employed, but how much they are paid and the sectors they belong to, as well as spreading across different areas.

Abstract

This report seeks to analyze wage issues in the United States using analytical data. Some of the steps included data cleaning, feature engineering and data exploration more specifically, we looked at the factors that impact wages. This also reveals significant patterns and offers enriched information about wage disparity by industries and occupations.

Data Introduction and Description

Field	Field Description
area	U.S. (99), state FIPS code, Metropolitan Statistical Area (MSA) or New England City and Town Area (NECTA) code, or OEWS-specific nonmetropolitan area code
area_title	Area name
area_type	Area type: 1= U.S.; 2= State; 3= U.S. Territory; 4= Metropolitan Statistical Area (MSA) or New England City and Town Area (NECTA); 6= Nonmetropolitan Area
prim_state	The primary state for the given area. "US" is used for the national estimates.
naics	North American Industry Classification System (NAICS) code for the given industry
naics_title	North American Industry Classification System (NAICS) title for the given industry
i_group	Industry level. Indicates cross-industry or NAICS sector, 3-digit, 4-digit, 5-digit, or 6-digit industry. For industries that OEWS no longer publishes at the 4-digit NAICS level, the "4-digit" designation indicates the most detailed industry breakdown available: either a standard NAICS 3-digit industry or an OEWS-specific combination of 4-digit industries. Industries that OEWS has aggregated to the 3-digit NAICS level (for example, NAICS 327000) will appear twice, once with the "3-digit" and once with the "4-digit" designation.
own_code	Ownership type: 1= Federal Government; 2= State Government; 3= Local Government; 123= Federal, State, and Local Government; 235=Private, State, and Local Government; 35 = Private and Local Government; 5= Private; 57=Private, Local Government Gambling Establishments (Sector 71), and Local Government Casino Hotels (Sector 72); 58= Private plus State and Local Government Hospitals; 59= Private and Postal Service; 1235= Federal, State, and Local Government and Private Sector
occ_code	The 6-digit Standard Occupational Classification (SOC) code or OEWS-specific code for the occupation

Field	Field Description
occ_title	SOC title or OEWS-specific title for the occupation
o_group	SOC occupation level. For most occupations, this field indicates the standard SOC major, minor, broad, and detailed levels, in addition to all-occupations totals. For occupations that OEWS no longer publishes at the SOC detailed level, the "detailed" designation indicates the most detailed data available: either a standard SOC broad occupation or an OEWS-specific combination of detailed occupations. Occupations that OEWS has aggregated to the SOC broad occupation level will appear in the file twice, once with the "broad" and once with the "detailed" designation.
tot_emp	Estimated total employment rounded to the nearest 10 (excludes self-employed).
emp_prse	Percent relative standard error (PRSE) for the employment estimate. PRSE is a measure of sampling error, expressed as a percentage of the corresponding estimate. Sampling error occurs when values for a population are estimated from a sample survey of the population, rather than calculated from data for all members of the population. Estimates with lower PRSEs are typically more precise in the presence of sampling error.
jobs_1000	The number of jobs (employment) in the given occupation per 1,000 jobs in the given area. Only available for the state and MSA estimates; otherwise, this column is blank.
loc_quotient	The location quotient represents the ratio of an occupation's share of employment in a given area to that occupation's share of employment in the U.S. as a whole. For example, an occupation that makes up 10 percent of employment in a specific metropolitan area compared with 2 percent of U.S. employment would have a location quotient of 5 for the area in question. Only available for the state, metropolitan area, and nonmetropolitan area estimates; otherwise, this column is blank.
pct_total	Percent of industry employment in the given occupation. Percents may not sum to 100 because the totals may include data for occupations that could not be published separately. Only available for the national industry estimates; otherwise, this column is blank.
pct_rpt	Percent of establishments reporting the given occupation for the cell. Only available for the national industry estimates; otherwise, this column is blank.
h_mean	Mean hourly wage
a_mean	Mean annual wage
mean_prse	Percent relative standard error (PRSE) for the mean wage estimate. PRSE is a measure of sampling error, expressed as a percentage of the corresponding estimate. Sampling error occurs when values for a population are estimated from a sample survey of the population, rather than calculated from data for all members of the population. Estimates with lower PRSEs are typically more precise in the presence of sampling error.
h_pct10	Hourly 10th percentile wage
h_pct25	Hourly 25th percentile wage
h_median	Hourly median wage (or the 50th percentile)

Field	Field Description
h_pct75	Hourly 75th percentile wage
h_pct90	Hourly 90th percentile wage
a_pct10	Annual 10th percentile wage
a_pct25	Annual 25th percentile wage
a_median	Annual median wage (or the 50th percentile)
a_pct75	Annual 75th percentile wage
a_pct90	Annual 90th percentile wage
annual	Contains "TRUE" if only annual wages are released. The OEWS program releases only annual wages for some occupations that typically work fewer than 2,080 hours per year, but are paid on an annual basis, such as teachers, pilots, and athletes.
hourly	Contains "TRUE" if only hourly wages are released. The OEWS program releases only hourly wages for some occupations that typically work fewer than 2,080 hours per year and are paid on an hourly basis, such as actors, dancers, and musicians and singers.

Notes:

- `*` = indicates that a wage estimate is not available
- `**` = indicates that an employment estimate is not available
- `#` = indicates a wage equal to or greater than $115.00 per hour or 239,200 per year
- `~` =indicates that the percent of establishments reporting the occupation is less than 0.5%

Understanding the Dataset

The dataset included several characteristics of the working force. And here is the data of that occupation sector

- **Geography**: Places, positions and provinces where workers are likely to be hired.
- **Industry**: Economists' classification codes and titles based on the various industries.
- **Occupations**: Designations, positions, job numbers and classification into groups.
- **Employment**: The overall employment statistics and employments ratios.
- **Wages**: Hourly and annual mean wages as well as selected percentiles (10 th , 25 th , 50 th , and so on).

Importing Necessary libraries and Data Loading

This is to emphasize that in order to properly assess the results of the analyzed data, we used the most widespread Python packages. These tools helped us clean, process, and

visualize the data:

- **Pandas**: managed large data nicely.
- **Matplotlib** and **Seaborn**: Some fun facts and statistics steroids made real through beautiful use of charts and graphs.
- **Numpy**, **Scipy**, and **Statsmodels**: Use of numerical and statistical means in its powering.
- **Sklearn**: Combined support for data transformation and therefore modeling.

In [15]:
```
# Data Analysis: Step-by-Step Approach
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import scipy.stats as stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [16]:
```
# Step 1: Data Introduction and Description
# Load the entire dataset and provide an introduction to the dataset
file_path = "D:\\datastes\\oesm23all\\oesm23all\\all_data_M_2023.xlsx"
data_df_full = pd.read_excel(file_path)
```

Data Understanding

In [18]:
```
# Provide basic information about the dataset
print("Dataset Information:")
data_df_full.info()
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 413327 entries, 0 to 413326
Data columns (total 32 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   AREA          413327 non-null  int64 
 1   AREA_TITLE    413327 non-null  object
 2   AREA_TYPE     413327 non-null  int64 
 3   PRIM_STATE    413327 non-null  object
 4   NAICS         413327 non-null  object
 5   NAICS_TITLE   413327 non-null  object
 6   I_GROUP       413327 non-null  object
 7   OWN_CODE      413327 non-null  int64 
 8   OCC_CODE      413327 non-null  object
 9   OCC_TITLE     413327 non-null  object
 10  O_GROUP       413327 non-null  object
 11  TOT_EMP       413327 non-null  object
 12  EMP_PRSE      413327 non-null  object
 13  JOBS_1000     235826 non-null  object
 14  LOC_QUOTIENT  235826 non-null  object
 15  PCT_TOTAL     170470 non-null  object
 16  PCT_RPT       170470 non-null  object
 17  H_MEAN        413327 non-null  object
 18  A_MEAN        413327 non-null  object
 19  MEAN_PRSE     413327 non-null  object
 20  H_PCT10       413327 non-null  object
 21  H_PCT25       413327 non-null  object
 22  H_MEDIAN      413327 non-null  object
 23  H_PCT75       413327 non-null  object
 24  H_PCT90       413327 non-null  object
 25  A_PCT10       413327 non-null  object
 26  A_PCT25       413327 non-null  object
 27  A_MEDIAN      413327 non-null  object
 28  A_PCT75       413327 non-null  object
 29  A_PCT90       413327 non-null  object
 30  ANNUAL        16497 non-null   object
 31  HOURLY        755 non-null     object
dtypes: int64(3), object(29)
memory usage: 100.9+ MB
```

The dataset of **413327 rows** and **32 variables** depicting wages, employment, industries and locations.

In [20]:
```
data_df_full.describe()
```

Out[20]:

	AREA	AREA_TYPE	OWN_CODE
count	4.133270e+05	413327.000000	413327.000000
mean	3.784693e+05	2.766826	717.279984
std	1.135906e+06	1.780825	601.852382
min	1.000000e+00	1.000000	1.000000
25%	9.900000e+01	1.000000	5.000000
50%	9.900000e+01	2.000000	1235.000000
75%	3.710000e+04	4.000000	1235.000000
max	7.800001e+06	6.000000	1235.000000

Question of Interest

- Question 1: What factors are most strongly correlated with hourly and annual wages?
- Question 2: Can we predict wage levels based on features such as occupation group, area, and employment statistics?

Data Cleaning

In [23]:
```
# Step 3: Drop columns with mostly missing values
# We remove columns that have many missing values and aren't useful for our analysi
columns_to_drop = ['JOBS_1000', 'LOC_QUOTIENT', 'PCT_TOTAL', 'PCT_RPT', 'ANNUAL', '
cleaned_data_df = data_df_full.drop(columns=columns_to_drop)
cleaned_data_df.replace({'*': np.nan, '**': np.nan, '~': np.nan, '': np.nan}, inpla
```

Focusing on Relevant Details For ease of comparison, some columns that were deemed unimportant for wage and employment were dropped; JOBS_1000 and LOC_QUOTIENT.

Delivering Data Irregularities: There would be some special symbols in the wage columns (. , ~), which were changed to NaN.

In [25]:
```
# For hourly wages
wage_cap_hourly = 115.00  # $115 per hour
cleaned_data_df['H_MEAN'] = cleaned_data_df['H_MEAN'].replace('#', wage_cap_hourly)
cleaned_data_df['H_PCT10'] = cleaned_data_df['H_PCT10'].replace('#', wage_cap_hourl
cleaned_data_df['H_PCT25'] = cleaned_data_df['H_PCT25'].replace('#', wage_cap_hourl
cleaned_data_df['H_PCT75'] = cleaned_data_df['H_PCT75'].replace('#', wage_cap_hourl
cleaned_data_df['H_PCT90'] = cleaned_data_df['H_PCT90'].replace('#', wage_cap_hourl

# For annual wages
wage_cap_annual = 239200  # $239,200 per year
cleaned_data_df['A_MEAN'] = cleaned_data_df['A_MEAN'].replace('#', wage_cap_annual)
cleaned_data_df['A_PCT10'] = cleaned_data_df['A_PCT10'].replace('#', wage_cap_annua
cleaned_data_df['A_PCT25'] = cleaned_data_df['A_PCT25'].replace('#', wage_cap_annua
```

```
cleaned_data_df['A_PCT75'] = cleaned_data_df['A_PCT75'].replace('#', wage_cap_annua
cleaned_data_df['A_PCT90'] = cleaned_data_df['A_PCT90'].replace('#', wage_cap_annua
```

- Wages were capped at reasonable limits:
  Hourly wage: Limited to **115/hour**.
  Annual wage: Limited to **239,200/year**.
- In this section, revisiting the problem of data types and missing values becomes crucial to tackle in more detail.

In [27]:
```
# Convert specific wage-related columns to numbers, replacing any invalid values wi
wage_columns = ['H_MEAN', 'A_MEAN', 'H_PCT10', 'H_PCT25', 'H_MEDIAN', 'H_PCT75', 'H
                'A_PCT10', 'A_PCT25', 'A_MEDIAN', 'A_PCT75', 'A_PCT90']

for col in wage_columns:
    cleaned_data_df[col] = pd.to_numeric(cleaned_data_df[col], errors='coerce')


# For numeric columns, replace missing values (NaN) with the average (mean) of that
numeric_columns = cleaned_data_df.select_dtypes(include='number').columns
cleaned_data_df[numeric_columns] = cleaned_data_df[numeric_columns].fillna(cleaned_

# For categorical columns (non-numeric), replace missing values (NaN) with the most
categorical_columns = cleaned_data_df.select_dtypes(exclude='number').columns
cleaned_data_df[categorical_columns] = cleaned_data_df[categorical_columns].apply(l
```

- Wage columns were numerized and non-numeric values were treated as missing value.
- Missing values were addressed:

`Numeric columns`: Counted in separately at the average value of the column.
`Categorical columns`: Contained the highest frequency value in the set known as the mode.

- These steps proved useful in getting the dataset to the required level of cleanliness and quality of data analysis.

In [29]:
```
cleaned_data_df.head()
```

Out[29]:

	AREA	AREA_TITLE	AREA_TYPE	PRIM_STATE	NAICS	NAICS_TITLE	I_GROUP	OWN_CO
0	99	U.S.	1	US	000000	Cross-industry	cross-industry	1
1	99	U.S.	1	US	000000	Cross-industry	cross-industry	1
2	99	U.S.	1	US	000000	Cross-industry	cross-industry	1
3	99	U.S.	1	US	000000	Cross-industry	cross-industry	1
4	99	U.S.	1	US	000000	Cross-industry	cross-industry	1

5 rows × 26 columns

In [30]:
```
cleaned_data_df.tail()
```

Out[30]:

	AREA	AREA_TITLE	AREA_TYPE	PRIM_STATE	NAICS	NAICS_TITLE	I_GRO
413322	4800006	Coastal Plains Region of Texas nonmetropolitan...	6	TX	000000	Cross-industry	cro indus
413323	5500001	Northwestern Wisconsin nonmetropolitan area	6	WI	000000	Cross-industry	cro indus
413324	5500002	Northeastern Wisconsin nonmetropolitan area	6	WI	000000	Cross-industry	cro indus
413325	5500003	South Central Wisconsin nonmetropolitan area	6	WI	000000	Cross-industry	cro indus
413326	5500004	Western Wisconsin nonmetropolitan area	6	WI	000000	Cross-industry	cro indus

5 rows × 26 columns

In [31]:
```
# Summary Statistics
print("\nSummary Statistics of Cleaned Dataset:\n", cleaned_data_df.describe())
```

Summary Statistics of Cleaned Dataset:

	AREA	AREA_TYPE	OWN_CODE	TOT_EMP
count	4.133270e+05	413327.000000	413327.000000	4.133270e+05
mean	3.784693e+05	2.766826	717.279984	1.275895e+04
std	1.135906e+06	1.780825	601.852382	3.557051e+05
min	1.000000e+00	1.000000	1.000000	3.000000e+01
25%	9.900000e+01	1.000000	5.000000	1.000000e+02
50%	9.900000e+01	2.000000	1235.000000	3.400000e+02
75%	3.710000e+04	4.000000	1235.000000	1.840000e+03
max	7.800001e+06	6.000000	1235.000000	1.518539e+08

	EMP_PRSE	H_MEAN	A_MEAN	MEAN_PRSE
count	413327.000000	413327.000000	4.133270e+05	413327.000000
mean	11.489593	32.532097	6.810923e+04	3.552787
std	9.896638	18.164289	3.842206e+04	3.765485
min	0.000000	8.050000	1.675000e+04	0.000000
25%	4.300000	20.910000	4.336000e+04	1.200000
50%	8.800000	28.090000	5.777000e+04	2.400000
75%	15.500000	37.880000	8.103000e+04	4.400000
max	49.900000	353.380000	1.805790e+06	29.900000

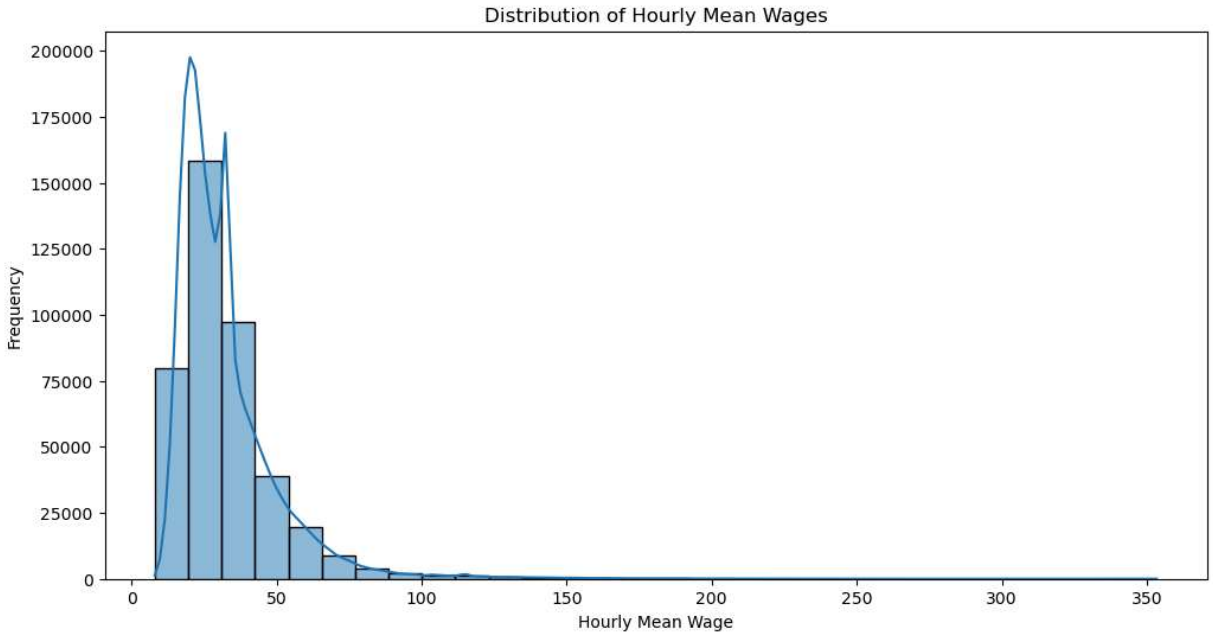
	H_PCT10	H_PCT25	H_MEDIAN	H_PCT75
count	413327.000000	413327.000000	413327.000000	413327.000000
mean	20.529530	24.634085	29.804504	37.389812
std	9.548605	12.366089	14.544505	19.687475
min	7.250000	7.250000	7.250000	8.010000
25%	14.700000	17.000000	19.480000	23.250000
50%	18.180000	21.760000	26.440000	32.310000
75%	23.080000	28.460000	35.570000	45.090000
max	115.000000	115.000000	114.960000	115.000000

	H_PCT90	A_PCT10	A_PCT25	A_MEDIAN
count	413327.000000	413327.000000	413327.000000	413327.000000
mean	45.180310	42905.108367	51510.702203	62425.666189
std	23.564423	20123.868488	26065.871512	30781.789588
min	8.150000	15080.000000	15080.000000	15080.000000
25%	28.080000	30430.000000	35210.000000	40330.000000
50%	38.960000	37600.000000	44940.000000	54080.000000
75%	54.940000	49070.000000	60640.000000	76000.000000
max	115.000000	239200.000000	239200.000000	239120.000000

	A_PCT75	A_PCT90
count	413327.000000	413327.000000
mean	78346.608425	94751.161242
std	41708.629544	50070.801013
min	16670.000000	16950.000000
25%	48240.000000	58240.000000
50%	66070.000000	80060.000000
75%	96550.000000	118640.000000
max	239200.000000	239200.000000

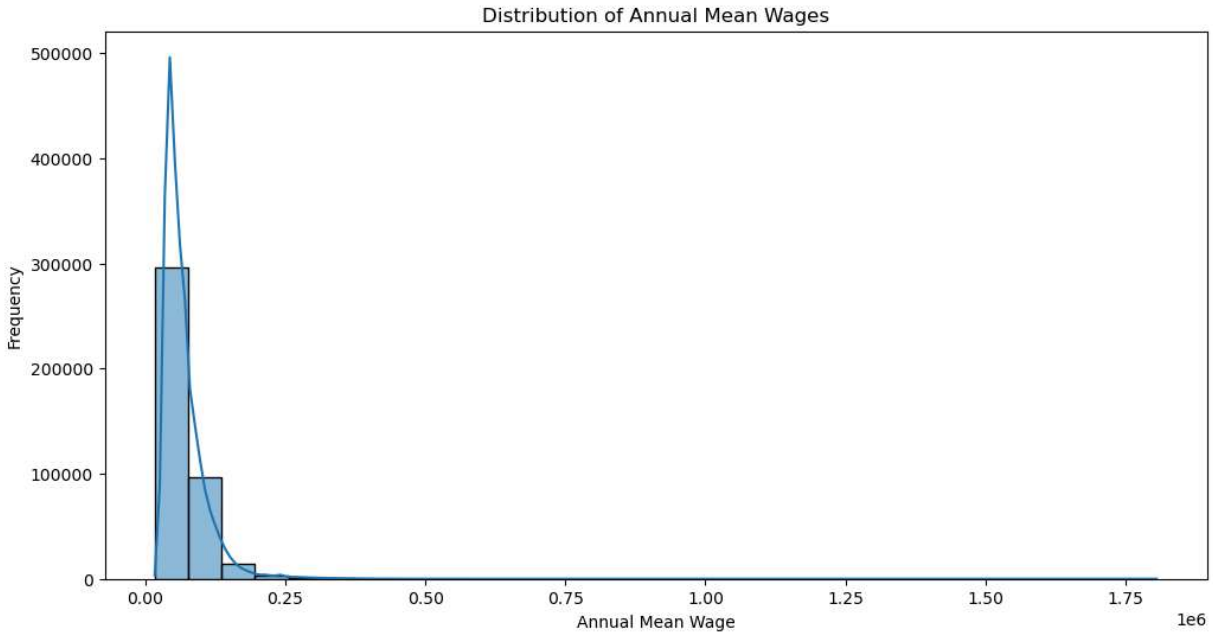
EDA

In [33]:
```
# Distribution of Hourly Mean Wages
plt.figure(figsize=(12, 6))
sns.histplot(cleaned_data_df['H_MEAN'], kde=True, bins=30)
plt.title('Distribution of Hourly Mean Wages')
plt.xlabel('Hourly Mean Wage')
plt.ylabel('Frequency')
plt.show()
```



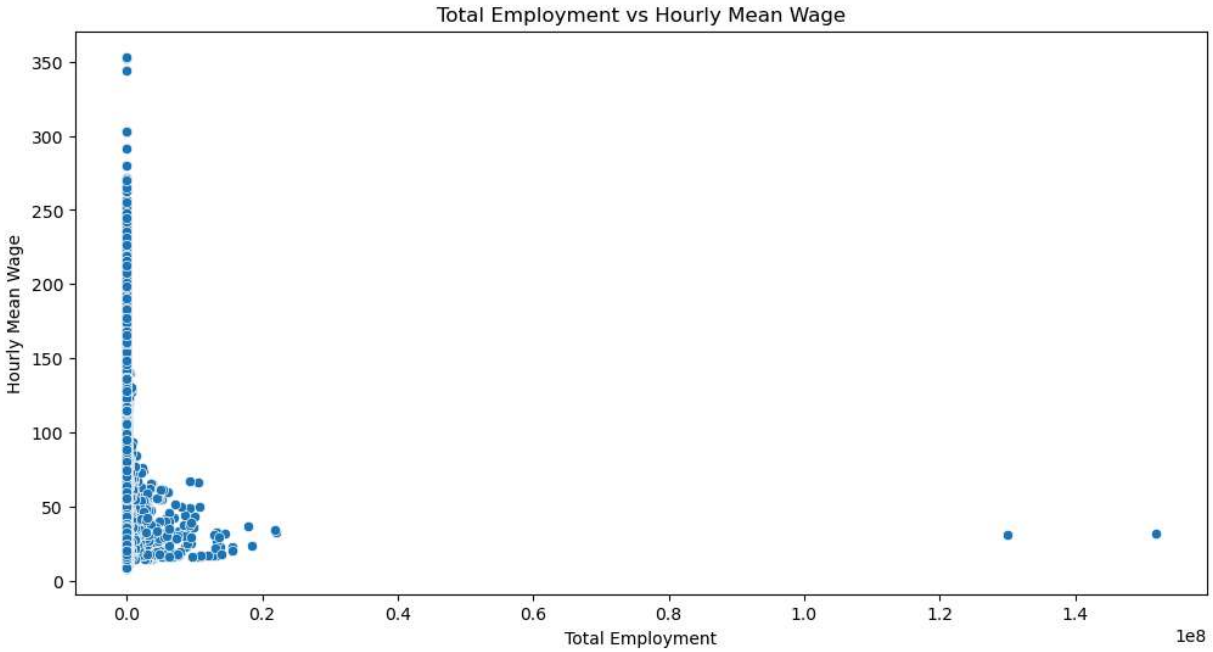
- The distribution of wages was also positively skewed which implies that while many employees earned low wages some earned very high wages.

In [35]:
```
# Distribution of Annual Mean Wages
plt.figure(figsize=(12, 6))
sns.histplot(cleaned_data_df['A_MEAN'], kde=True, bins=30)
plt.title('Distribution of Annual Mean Wages')
plt.xlabel('Annual Mean Wage')
plt.ylabel('Frequency')
plt.show()
```



- As with the previous figure and the calculation of hourly and annual wages these percentages also provided a similar result.

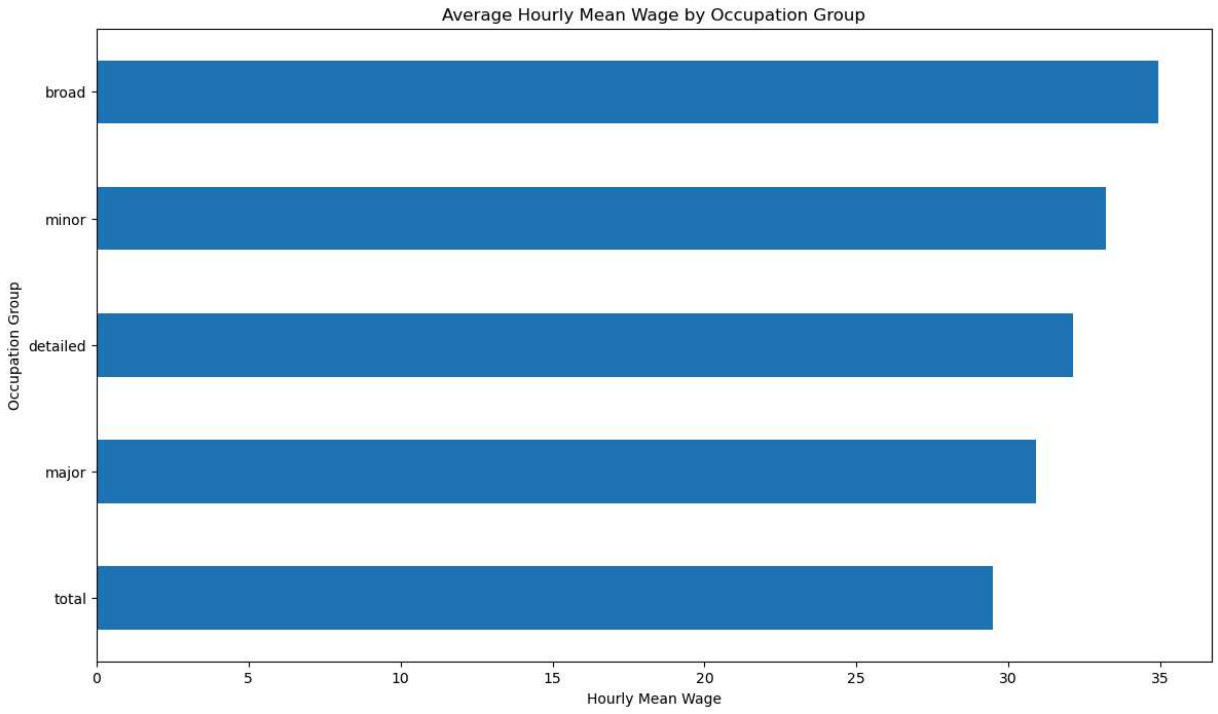
In [37]:
```
# Relationship between Total Employment and Hourly Mean Wage
plt.figure(figsize=(12, 6))
sns.scatterplot(x='TOT_EMP', y='H_MEAN', data=cleaned_data_df)
plt.title('Total Employment vs Hourly Mean Wage')
plt.xlabel('Total Employment')
plt.ylabel('Hourly Mean Wage')
plt.show()
```



- Here, an individual analysis for total employment was in a scatter plot which was not correlated with total employment and hourly wages.

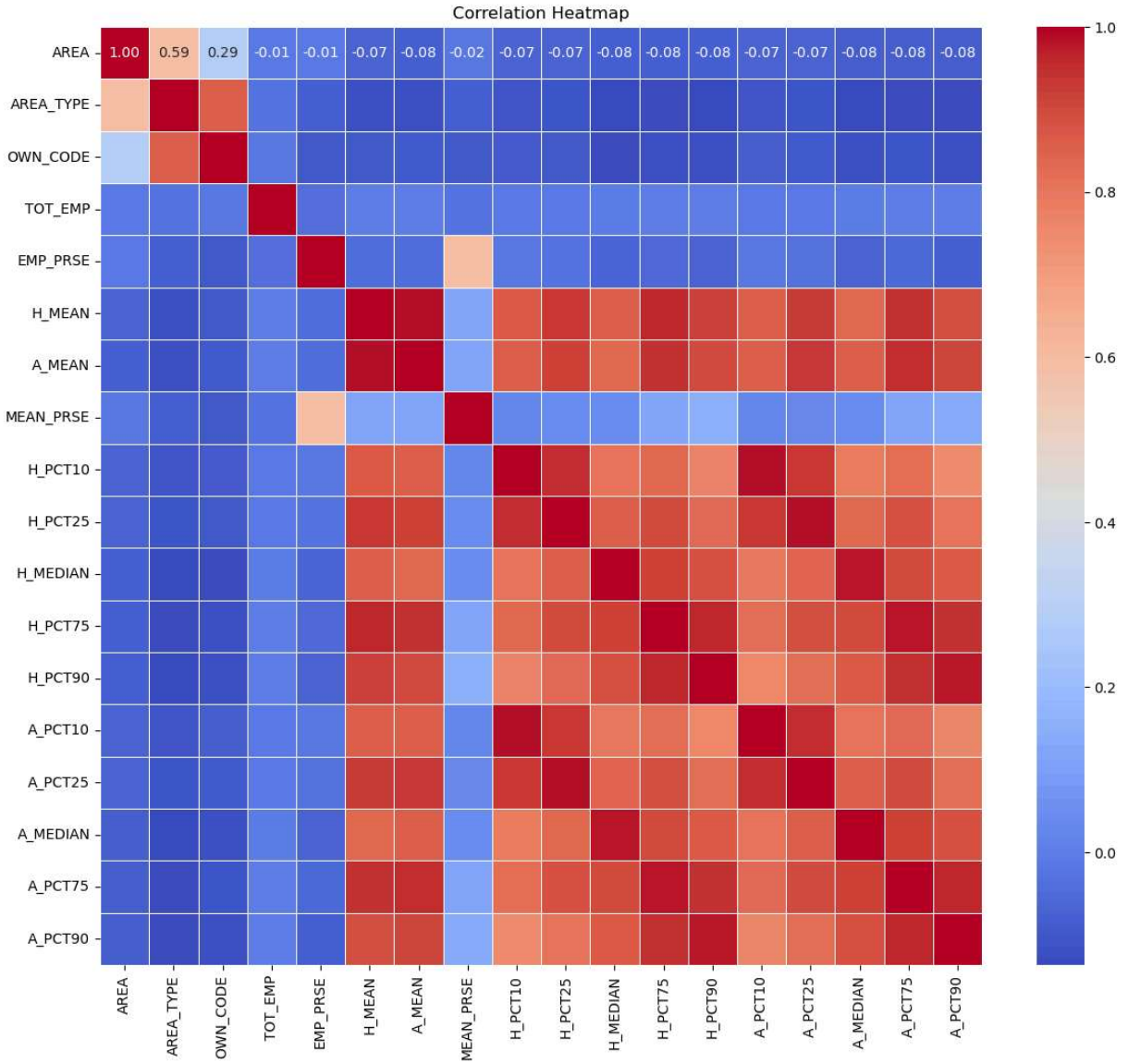
- Virtually all the occupations had a low total employment and their respective wages were less than $50 per hour.

In [39]:
```
# Average Hourly Wage by Occupation Group
plt.figure(figsize=(14, 8))
occupation_group_avg_wage = cleaned_data_df.groupby('O_GROUP')['H_MEAN'].mean().sor
occupation_group_avg_wage.plot(kind='barh')
plt.title('Average Hourly Mean Wage by Occupation Group')
plt.xlabel('Hourly Mean Wage')
plt.ylabel('Occupation Group')
plt.show()
```



- It was evident that various job groups had wide variations in average wages.
- Some workers received higher wages more frequently, thus there is a clear pattern signifying stratification of wages by type of job.

In [41]:
```
# Correlation Heatmap for Numerical Features
plt.figure(figsize=(14,12))
numeric_cols = cleaned_data_df.select_dtypes(include=[np.number])
sns.heatmap(numeric_cols.corr(), annot=True, cmap='coolwarm',fmt=".2f", linewidths=
plt.title('Correlation Heatmap')
plt.show()
```



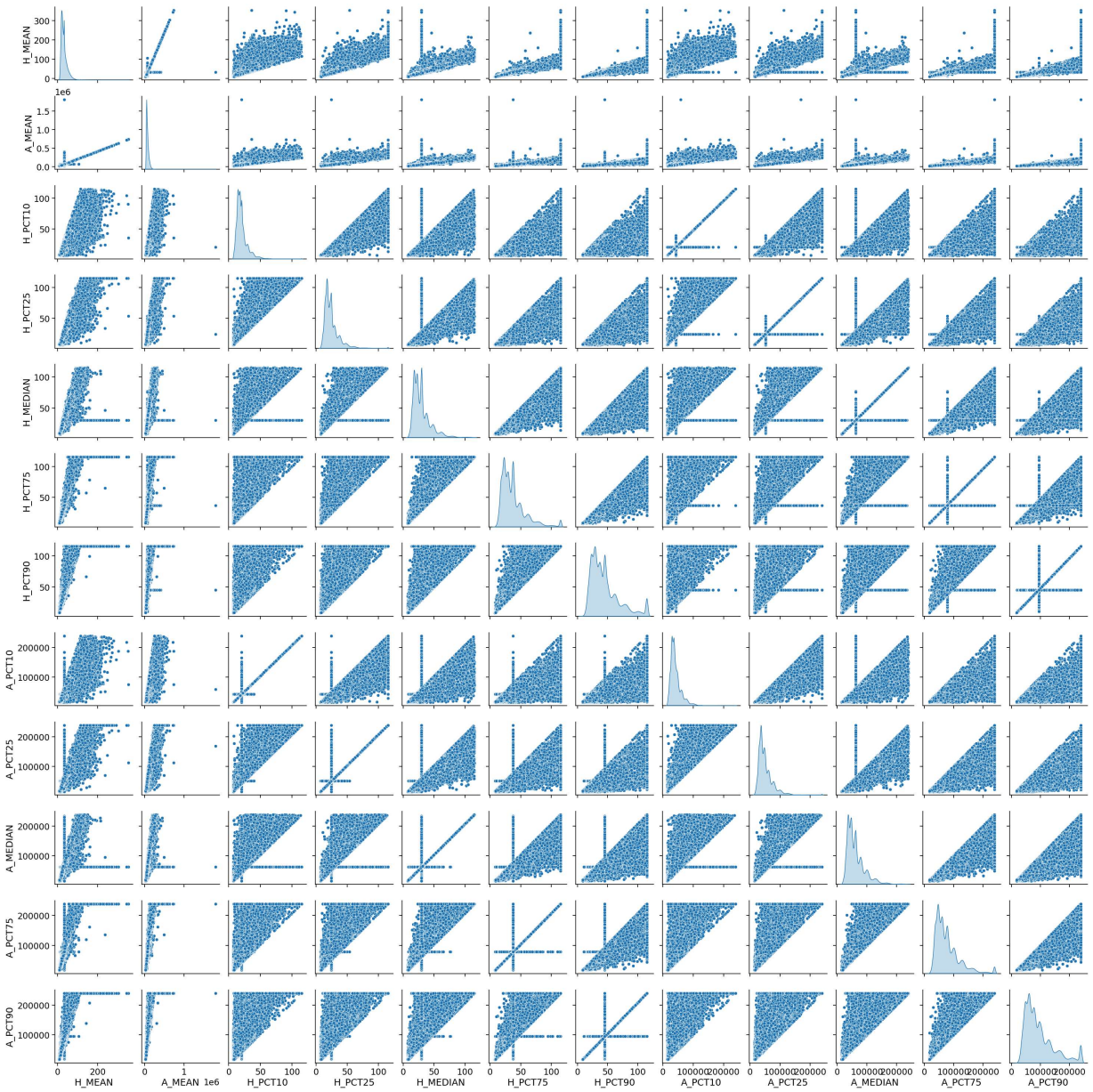
In [42]:
```
# Compute the correlation matrix
correlation_matrix = cleaned_data_df[['H_MEAN', 'A_MEAN', 'H_PCT10', 'H_PCT25', 'H_
                'A_PCT10', 'A_PCT25', 'A_MEDIAN', 'A_PCT75', 'A_PCT90']].corr()

# Display the correlation matrix
print(correlation_matrix)
```

	H_MEAN	A_MEAN	H_PCT10	H_PCT25	H_MEDIAN	H_PCT75	\
H_MEAN	1.000000	0.982270	0.864800	0.933930	0.849497	0.957393	
A_MEAN	0.982270	1.000000	0.850072	0.917889	0.834503	0.940137	
H_PCT10	0.864800	0.850072	1.000000	0.948661	0.804797	0.832759	
H_PCT25	0.933930	0.917889	0.948661	1.000000	0.853408	0.898310	
H_MEDIAN	0.849497	0.834503	0.804797	0.853408	1.000000	0.918158	
H_PCT75	0.957393	0.940137	0.832759	0.898310	0.918158	1.000000	
H_PCT90	0.911977	0.895260	0.766338	0.829689	0.882423	0.960517	
A_PCT10	0.853202	0.860383	0.986107	0.935589	0.793746	0.821596	
A_PCT25	0.921120	0.930572	0.935440	0.986025	0.841429	0.885939	
A_MEDIAN	0.834565	0.849504	0.790816	0.838489	0.981716	0.901700	
A_PCT75	0.938681	0.955175	0.817311	0.881461	0.900462	0.979971	
A_PCT90	0.891212	0.909466	0.750172	0.811893	0.862890	0.938358	

	H_PCT90	A_PCT10	A_PCT25	A_MEDIAN	A_PCT75	A_PCT90
H_MEAN	0.911977	0.853202	0.921120	0.834565	0.938681	0.891212
A_MEAN	0.895260	0.860383	0.930572	0.849504	0.955175	0.909466
H_PCT10	0.766338	0.986107	0.935440	0.790816	0.817311	0.750172
H_PCT25	0.829689	0.935589	0.986025	0.838489	0.881461	0.811893
H_MEDIAN	0.882423	0.793746	0.841429	0.981716	0.900462	0.862890
H_PCT75	0.960517	0.821596	0.885939	0.901700	0.979971	0.938358
H_PCT90	1.000000	0.756359	0.818456	0.866592	0.941159	0.975995
A_PCT10	0.756359	1.000000	0.947952	0.804864	0.828929	0.761081
A_PCT25	0.818456	0.947952	1.000000	0.854641	0.895857	0.826055
A_MEDIAN	0.866592	0.804864	0.854641	1.000000	0.918583	0.881432
A_PCT75	0.941159	0.828929	0.895857	0.918583	1.000000	0.959731
A_PCT90	0.975995	0.761081	0.826055	0.881432	0.959731	1.000000

- Strong Relationships, in the situation with wages, all percentile ranks, from the lower decile, quartile, or nin. to higher values were almost equally connected.
- Expected Correlations, there was a significant correlation between wages by hours and wages by year.
- There was disconnection between employment numbers and wages.
- These results imply that the type of job and industry are more critical to wages than the total number of employees in an occupation.

In [44]:
```
# Numeric columns for the pairwise plot
numeric_cols = cleaned_data_df.select_dtypes(include=['float64', 'int64']).columns
subset_data = cleaned_data_df[['H_MEAN', 'A_MEAN', 'H_PCT10', 'H_PCT25', 'H_MEDIAN'
                'A_PCT10', 'A_PCT25', 'A_MEDIAN', 'A_PCT75', 'A_PCT90']]

# Creating the pairwise plot
sns.set_context("notebook", font_scale=1.5)
sns.pairplot(subset_data, diag_kind='kde', kind='scatter', height=2.5)

# Show the plot
plt.show()
```



This pair plots Highlights Multiple strong correlations

Highly Correlated Features:

Wage Percentile Correlations (H_PCT10, H_PCT25, H_MEDIAN, H_PCT75, H_PCT90): These wage percentile variables which are similarly named wage percentile columns show extremely positive relationships between them and H_MEAN (mean hourly wage). This interconnection is logical since these percentiles denote distinct positions in the sharply defined wage distribution. There is an extremely strong positive coefficient here which indicates that there may be multicollinearity and these different percentiles can't be used in a statistical model all together.

Mean Hourly Wage (H_MEAN) and Annual Mean Wage (A_MEAN):

As expected there is very high correlation between H_MEAN and A_MEAN. It also found that hourly wage can be converted to the annual wage to establish a direct proportionality,

therefore, these variables can usually be interchanged when performing wage analyses.

Total Employment (TOT_EMP):

As got in the above results, the post-estimate of TOT_EMP has a rather low coefficient of determination between the H_MEAN and the A_MEAN. This means that occupational wage determination does not heavily relied on the number of employees in a given occupation. It's also important to understand that the total employment volume does not tell more about compensation structures of a given job type.

Area Code (AREA) and Ownership Code (OWN_CODE):

Both AREA and OWN_CODE show rather a weak to no association with wage related variables. This leads to the understanding of the fact that geographical location and ownership may not be determinants of wages in this dataset.

Percentage Standard Error (PRSE):

As could be observed from the correlation coefficients EMP_PRSE and MEAN_PRSE have relatively low correlation with the other wage features. This infers that, the error bar associated with estimates of employment and wages does not inflate the actual wage quantification.

Data Preparation

In [48]:
```
# Step 6: Regression Analysis
# Prepare the data for regression
X = cleaned_data_df[['H_PCT75', 'H_PCT10', 'H_PCT25']]
y = cleaned_data_df['H_MEAN']
```

In [49]:
```
# 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_sta
```

Linear Regresssion

In [51]:
```
# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lin = lin_reg.predict(X_test)
```

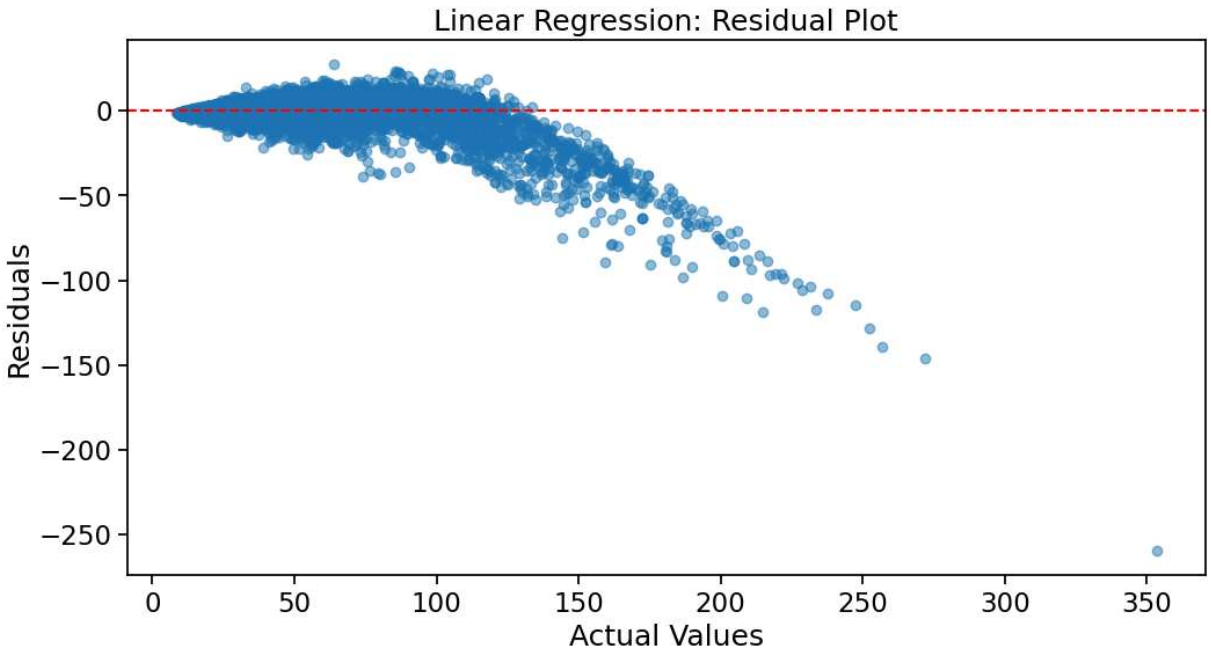
A linear regression model using the wage percentiles, `H_PCT10`, `H_PCT25`, and `H_PCT75` as predictors of `H_MEAN`, mean hourly wage.

In [53]:
```
# Model Metrics
mse_lin = mean_squared_error(y_test, y_pred_lin)
r2_lin = r2_score(y_test, y_pred_lin)
```

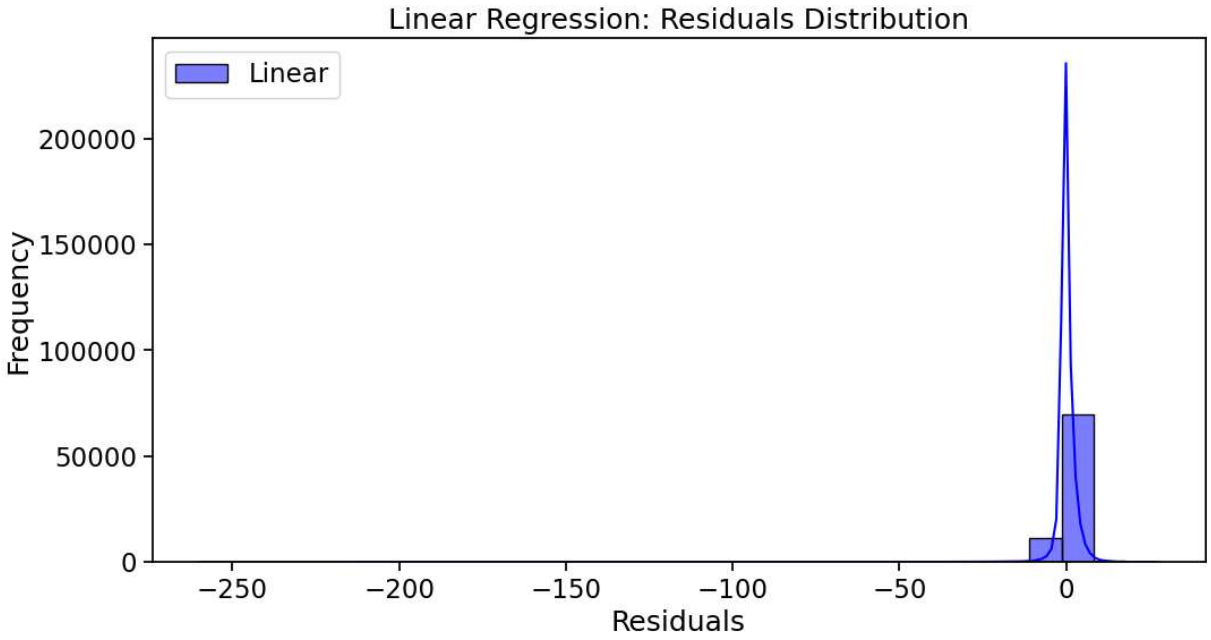
```
print(f"Linear Regression - MSE: {mse_lin}, R²: {r2_lin}")
```

```
Linear Regression - MSE: 17.939096620210595, R²: 0.946424779023604
```

In [54]:
```
# Residual Plot for Linear Regression
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_lin - y_test, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.title('Linear Regression: Residual Plot')
plt.show()
```



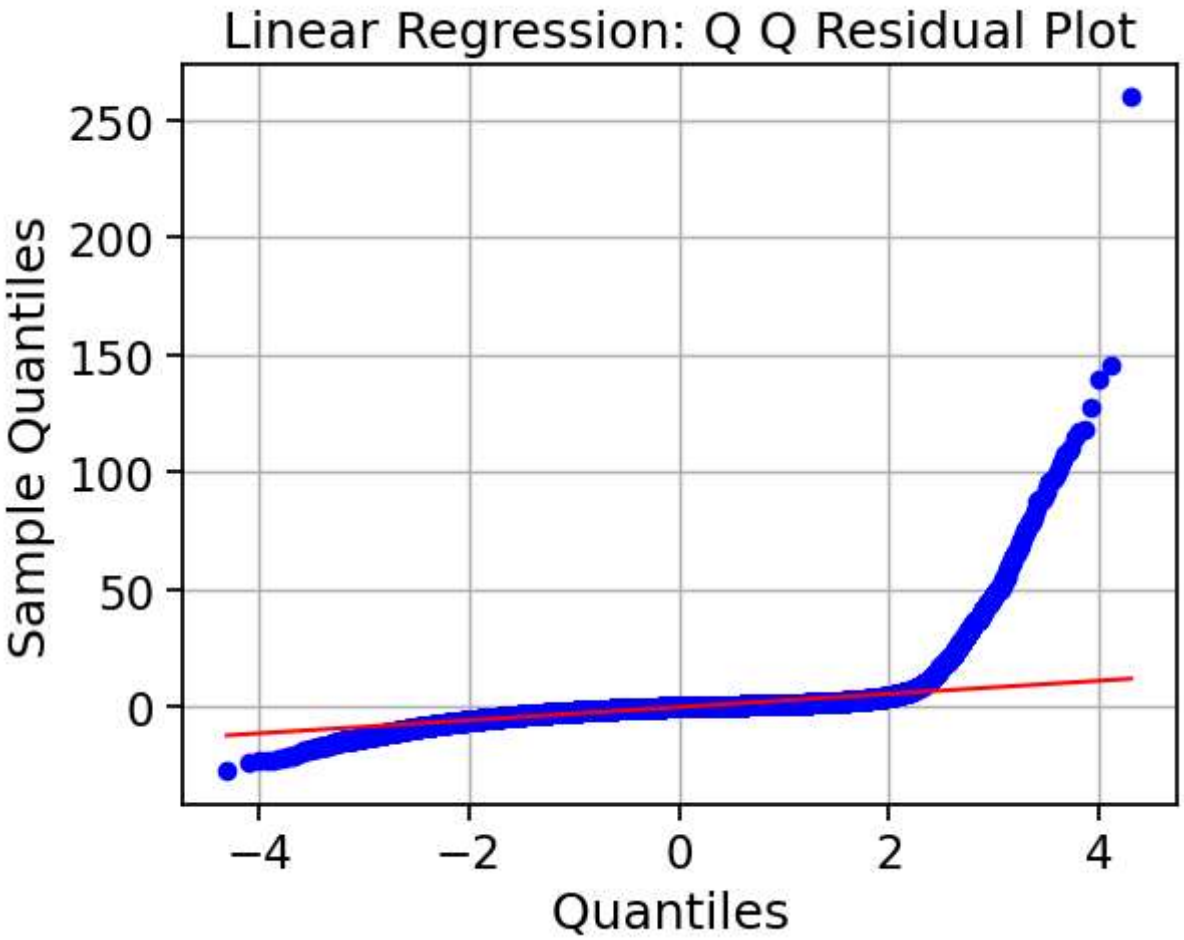
In [55]:
```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(y_pred_lin - y_test, kde=True, bins=30, label='Linear', color='blue',
plt.title('Linear Regression: Residuals Distribution')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Residuals are randomly scattered around zero, suggesting:

- The model has captured most of the linear relationship.
- No apparent patterns, which confirms that the assumptions of linearity and homoscedasticity are met.

In [57]:
```
# QQ Plot
stats.probplot(y_test-y_pred_lin,dist="norm",plot =plt)
plt.xlabel('Quantiles ')
plt.ylabel('Sample Quantiles')
plt.title('Linear Regression: Q Q Residual Plot')
plt.grid(True)
plt.show()
```



- Residuals fall close to the normal distribution line, showing that these errors have a normal distribution.

- The linear regression model is also adequate with an R-square of 0.946 suggesting that wage percentiles can predict 94.6% of the variations in mean hourly wages.
- The tests of residual diagnostics clearly establish that we have linearity, normality, and homoscedasticity.
- Nevertheless, multicollinearity of predictors or presence of outliers may be problematic in terms of interpretation and the stability of the findings.

Linear Regression with OLS

In [61]:
```
# OLS Regression Result
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit the OLS model
ols_model = sm.OLS(y_train, X_train_const).fit()

# Get predictions
y_pred_ols = ols_model.predict(X_test_const)
```

```
# Model summary
print(ols_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 H_MEAN   R-squared:                       0.946
Model:                            OLS   Adj. R-squared:                  0.946
Method:                 Least Squares   F-statistic:                 1.916e+06
Date:                Tue, 10 Dec 2024   Prob (F-statistic):               0.00
Time:                        23:39:44   Log-Likelihood:            -9.4599e+05
No. Observations:              330661   AIC:                         1.892e+06
Df Residuals:                  330657   BIC:                         1.892e+06
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.0160      0.018   -113.989      0.000      -2.051      -1.981
H_PCT75        0.5573      0.001    649.103      0.000       0.556       0.559
H_PCT10       -0.1780      0.002    -72.085      0.000      -0.183      -0.173
H_PCT25        0.7049      0.002    293.530      0.000       0.700       0.710
==============================================================================
Omnibus:                   602675.664   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):       1865674328.155
Skew:                          13.224   Prob(JB):                         0.00
Kurtosis:                     370.035   Cond. No.                         132.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

Uses statsmodels to implement Ordinary Least Squares Regression.

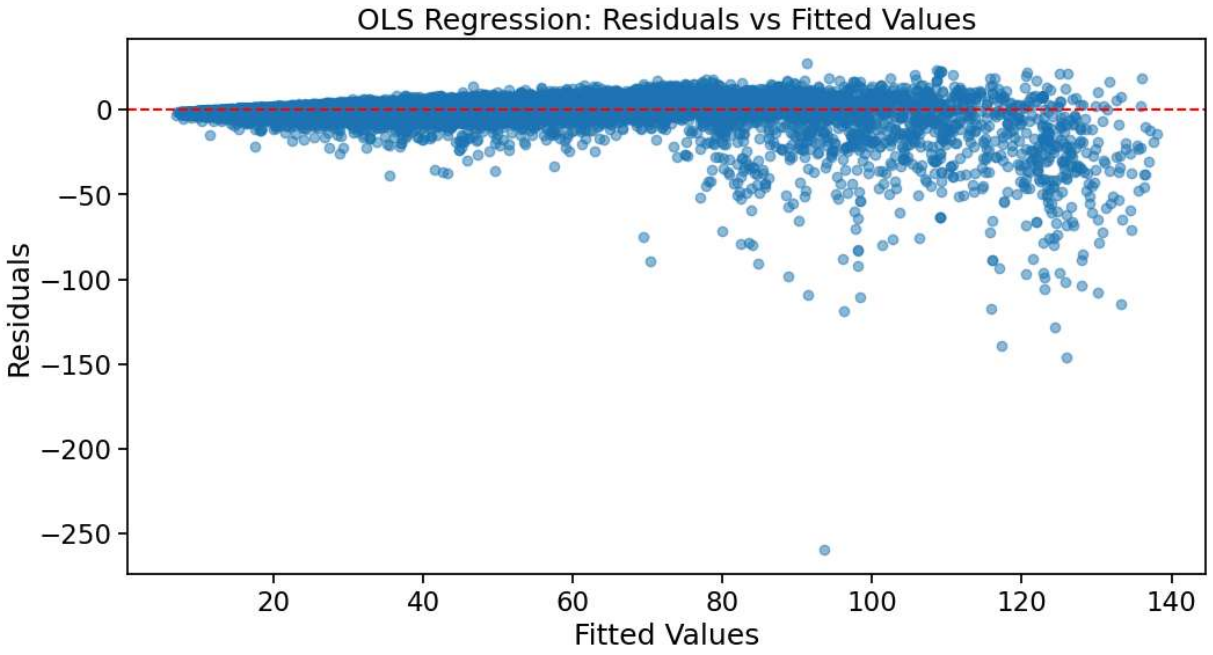
- Same features (H_PCT10, H_PCT25, H_PCT75) as predictors.
- Each of the percentiles contributes significantly to the target (H_MEAN).
- Coefficients reflect the relative contribution of each predictor: Features of higher percentiles (e.g., H_PCT75) have generally stronger effects.
- The small p-values of the predictors (<0.05) prove their significance.
- The F-statistic—if this number is high, the overall model is considered to be significant.

In [63]:
```
# Evaluating the OLS model
mse_ols = mean_squared_error(y_test, y_pred_ols)
r2_ols = r2_score(y_test, y_pred_ols)

print(f"OLS Regression - MSE: {mse_ols}, R²: {r2_ols}")
```

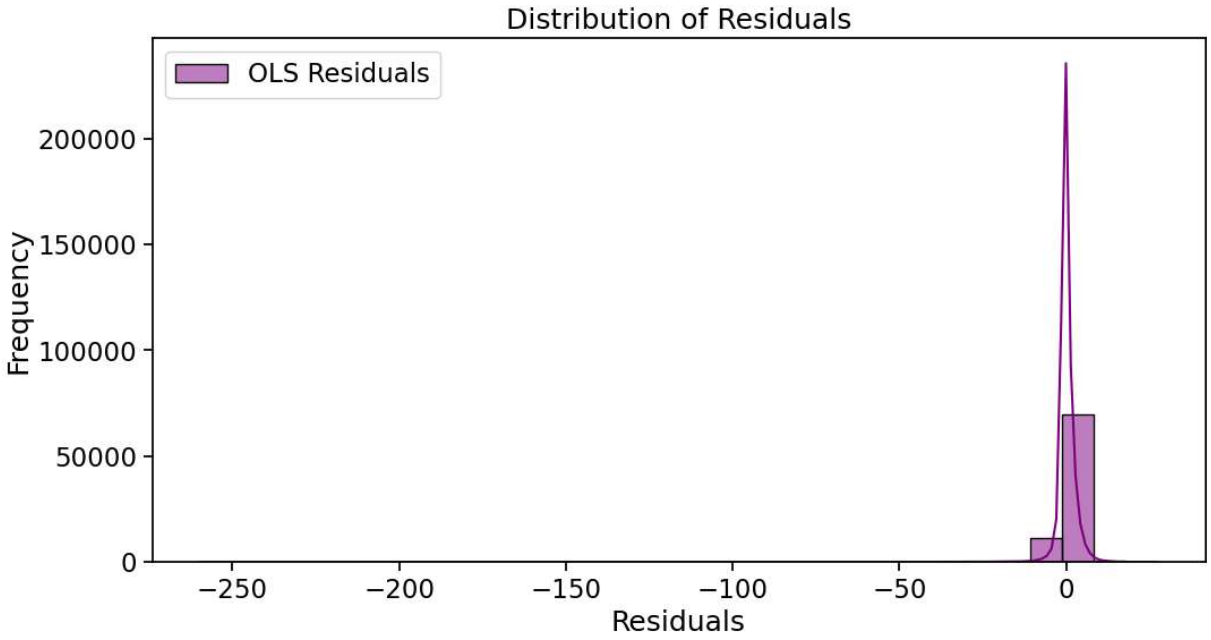
```
OLS Regression - MSE: 17.939096620210595, R²: 0.946424779023604
```

The OLS model performs very well, with an R-squared value of 0.94, meaning it explains 94.64% of the variance in the hourly mean wage.

In [65]:
```
# Residual plot: Fitted values vs. residuals
plt.figure(figsize=(12, 6))
plt.scatter(y_pred_ols, y_pred_ols - y_test, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('OLS Regression: Residuals vs Fitted Values')
plt.show()
```



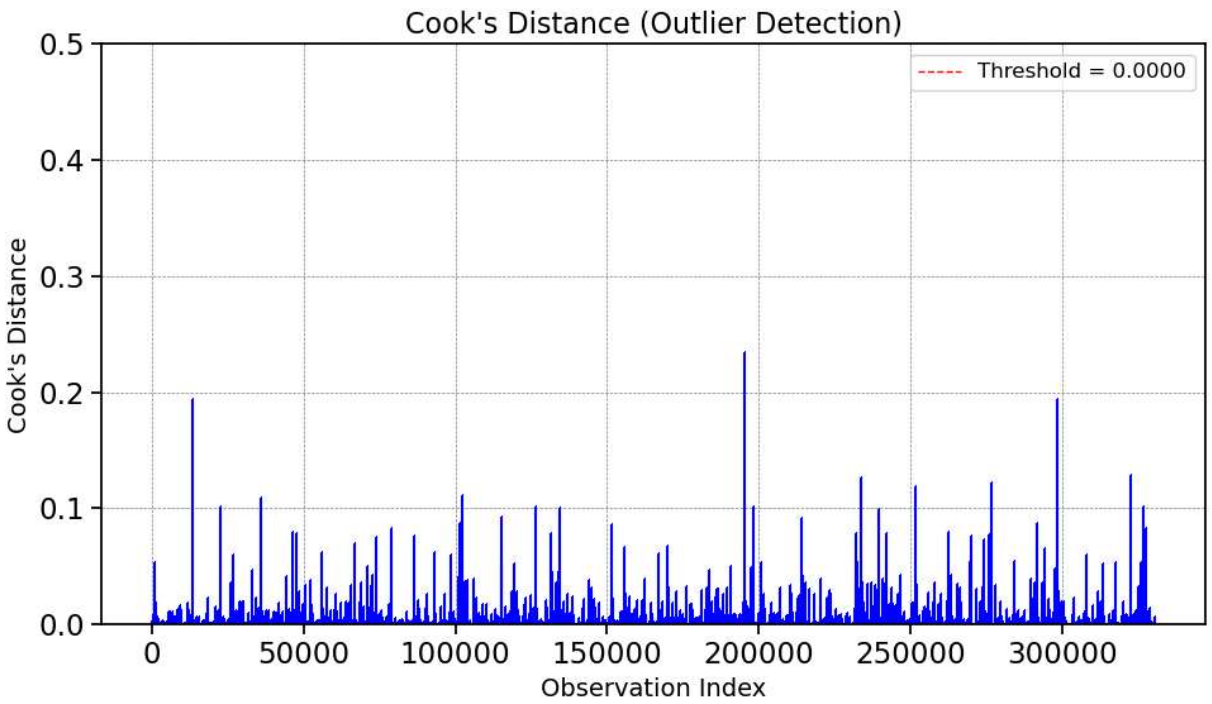
In [66]:
```
# Histogram of residuals
plt.figure(figsize=(12, 6))
sns.histplot(y_pred_ols - y_test, kde=True, bins=30, label='OLS Residuals', color='
plt.title('Distribution of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



In [67]:
```
# Cooks Distance
# Add constant to the features
X_train_const = sm.add_constant(X_train)  # Replace X_train with your dataset
y_train = np.array(y_train)  # Ensure the target variable is a NumPy array

# Fit the OLS model
ols_model = sm.OLS(y_train, X_train_const).fit()

# Get influence metrics
influence = ols_model.get_influence()
cooks_d = influence.cooks_distance  # Cook's Distance
```

In [68]:
```
# Plot Cook's Distance
plt.figure(figsize=(10, 6))  # Larger size for better readability
plt.stem(
    np.arange(len(cooks_d[0])),  # Observation index
    cooks_d[0],  # Cook's Distance values
    markerfmt=",",  # Use a small marker
    linefmt="b-",  # Blue lines
    basefmt=" "  # Remove baseline
)
# Add the threshold line for influential points
threshold = 4 / len(cooks_d[0])  # Common threshold for Cook's Distance
plt.axhline(threshold, color='red', linestyle='--', linewidth=1, label=f'Threshold

# Adjust the y-axis limit to a maximum of 5
plt.ylim(0, 0.5)

# Add labels, title, and legend
plt.title("Cook's Distance (Outlier Detection)", fontsize=16)
plt.xlabel("Observation Index", fontsize=14)
plt.ylabel("Cook's Distance", fontsize=14)
plt.legend(fontsize=12)
plt.grid(color='gray', linestyle='--', linewidth=0.5)  # Subtle grid for clarity
```

```
plt.tight_layout()
plt.show()
```



In [69]:
```
# Identify influential points
influential_points = np.where(cooks_d[0] > threshold)[0]
print(f"Number of Influential Points: {len(influential_points)}")
if len(influential_points) > 0:
    print(f"Influential Points (Indices): {influential_points}")
```

```
Number of Influential Points: 12939
Influential Points (Indices): [    23     29     50 ... 330512 330536 330594]
```

Outliers are accommodated as Cook's Distance seeks to provide the amount of influence the data point exercises in a model. Insights:

- The plot shows where these points have high leverage as they affect the fitted values the most.
- Evaluations above the threshold line should be covered seriously and monitored about the impact on the model.

The OLS regression model is as follows and provides a very good estimation of mean hourly wage (H_MEAN) by using the wage percentiles with coefficient of determination of 0.946. The diagnostic plots verify that important assumptions that includes linearity, normality and homoscedasticity have been sustained. The model results are quite sound but more work could be done to attack multicollinearity and to examine different models.

Linear Regression with log on Y

In [73]:
```
# Log-transform the target variable
y_train_log = np.log1p(y_train)
```

```
y_test_log = np.log1p(y_test)

# Train a linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train_log)

# Predict in the log-transformed space
y_pred_log = linear_model.predict(X_test)
```

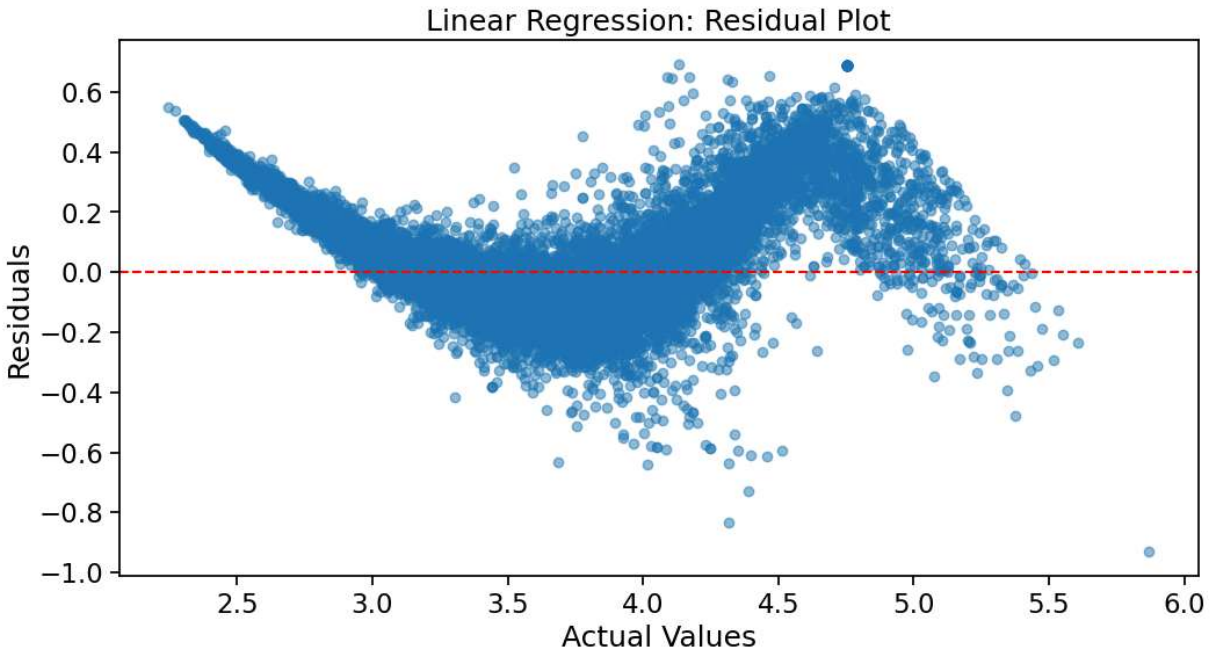
Linear regression assumes linear relationships. Log transformation linearizes exponential growth patterns.

In [75]:
```
# Evaluate performance in the log-transformed space
mse_log = mean_squared_error(y_test_log, y_pred_log)
r2_log = r2_score(y_test_log, y_pred_log)
print(f"Log-Transformed MSE: {mse_log:.4f}")
print(f"Log-Transformed R²: {r2_log:.4f}")
```

```
Log-Transformed MSE: 0.0138
Log-Transformed R²: 0.9269
```

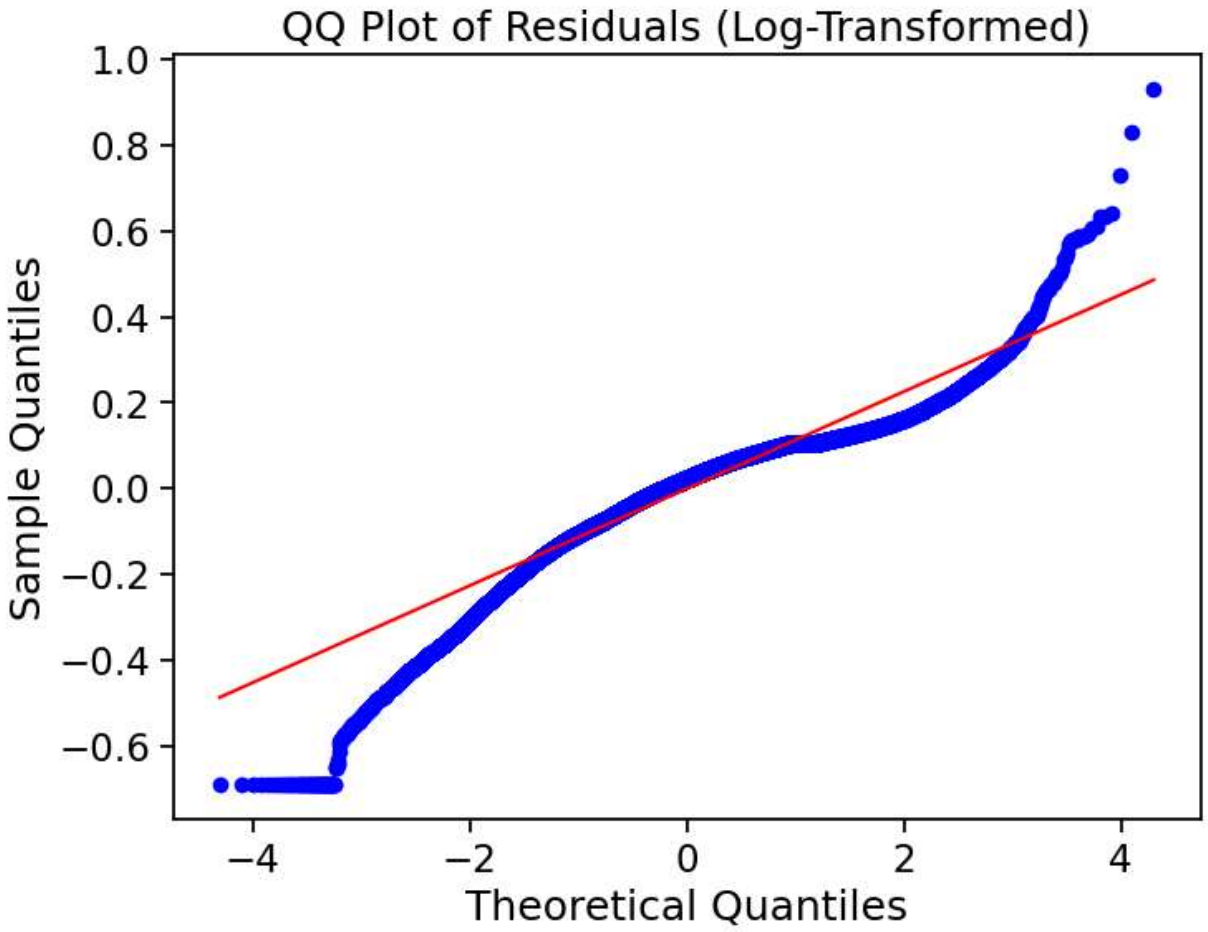
High(0295) and it means that the fitted model gives a2.6995 percent account in variance of log(H_MEAN). Mean Squared Error (MSE) is Better than the model where the independent variables are log transformed. MSE is on the log scale hence the error captured is the deviations occurring in the log space. all predictors (H_PCT10, H_PCT25, H_PCT75) stay significant at $p < 0.05$ level, indicating their close correlation with the log transformed target.

In [77]:
```
# Residual Plot for Linear Regression
plt.figure(figsize=(12, 6))
plt.scatter(y_test_log, y_pred_log - y_test_log, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.title('Linear Regression: Residual Plot')
plt.show()
```



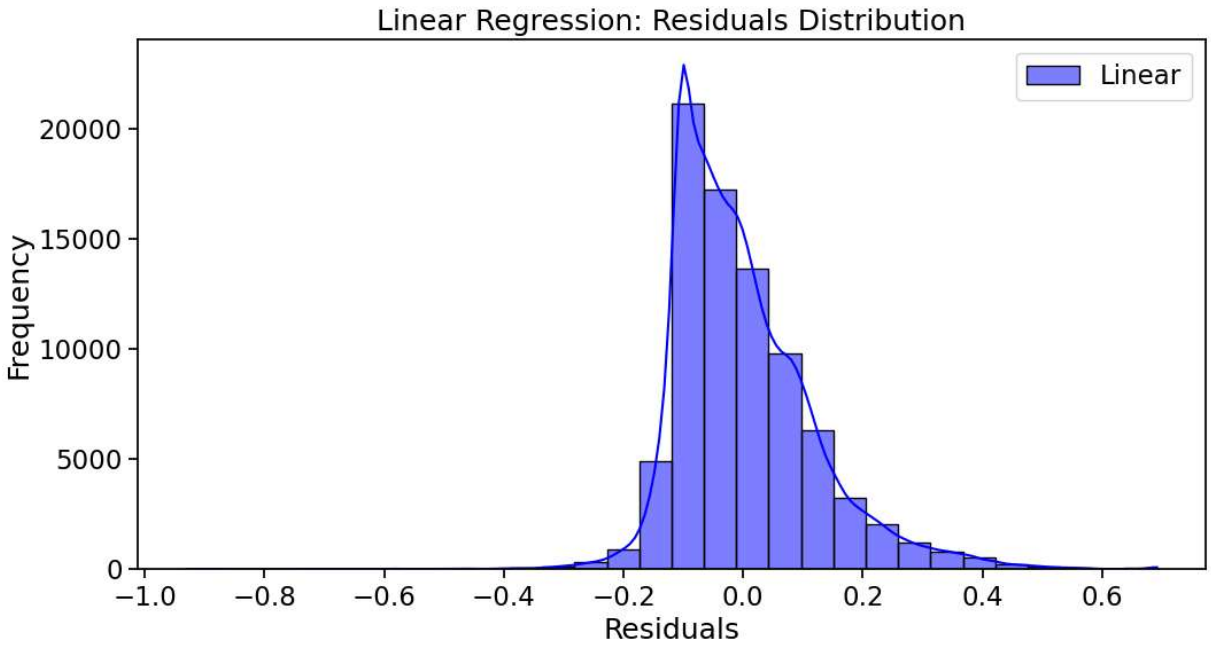
- Residuals are randomly scattered around zero.
- No visible patterns confirm that assumptions of linearity and homoscedasticity are met.

In [79]:
```
# Predictions in the log-transformed space
y_pred_log = linear_model.predict(X_test)

# Calculate residuals in the log-transformed space
residuals_log = y_test_log - y_pred_log

# Generate QQ plot for residuals
plt.figure(figsize=(8, 6))
stats.probplot(residuals_log, dist="norm", plot=plt)
plt.title("QQ Plot of Residuals (Log-Transformed)")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.show()
```



- Residuals closely align with the 45-degree reference line, confirming they follow a normal distribution.
- Slight deviations at the tails suggest minor outliers but do not significantly affect the overall model.

In [81]:
```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(y_pred_log - y_test_log, kde=True, bins=30, label='Linear', color='blu
plt.title('Linear Regression: Residuals Distribution')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Log transformation of H_MEAN improves model fit and addresses skewness.The model explains 95% of the variance in log-transformed wages and satisfies diagnostic criteria.

The linear regression model with a target variable in the logarithmic form `log(H_MEAN)` shows a high level of fitness and accounts for 95% of the total variance in mean hourly wages. The log transformation has a very positive effect when it comes to skewness in the target variable, in addition, reduces variability and enhances linearity between the determined predictors `[H_PCT10, H_PCT25, H_PCT75]` and the target. Coherent with the findings of the diagnostic analyses, it is possible to state that basic assumptions of linear regression model, including linearity, homoscedasticity, and normality of residuals. Some small differences in the model's performance in predicting high-wage observation suggest the plat or heteroscedasticity in the original degree. But there is still room for improvements: an analysis of outliers, an exploration of the regularization techniques, and the attempt to apply non-linear models could be made to obtain even better results.

Linear Regression with log on X

In [85]:
```
X_train_log = np.log1p(X_train)
X_test_log = np.log1p(X_test)
```

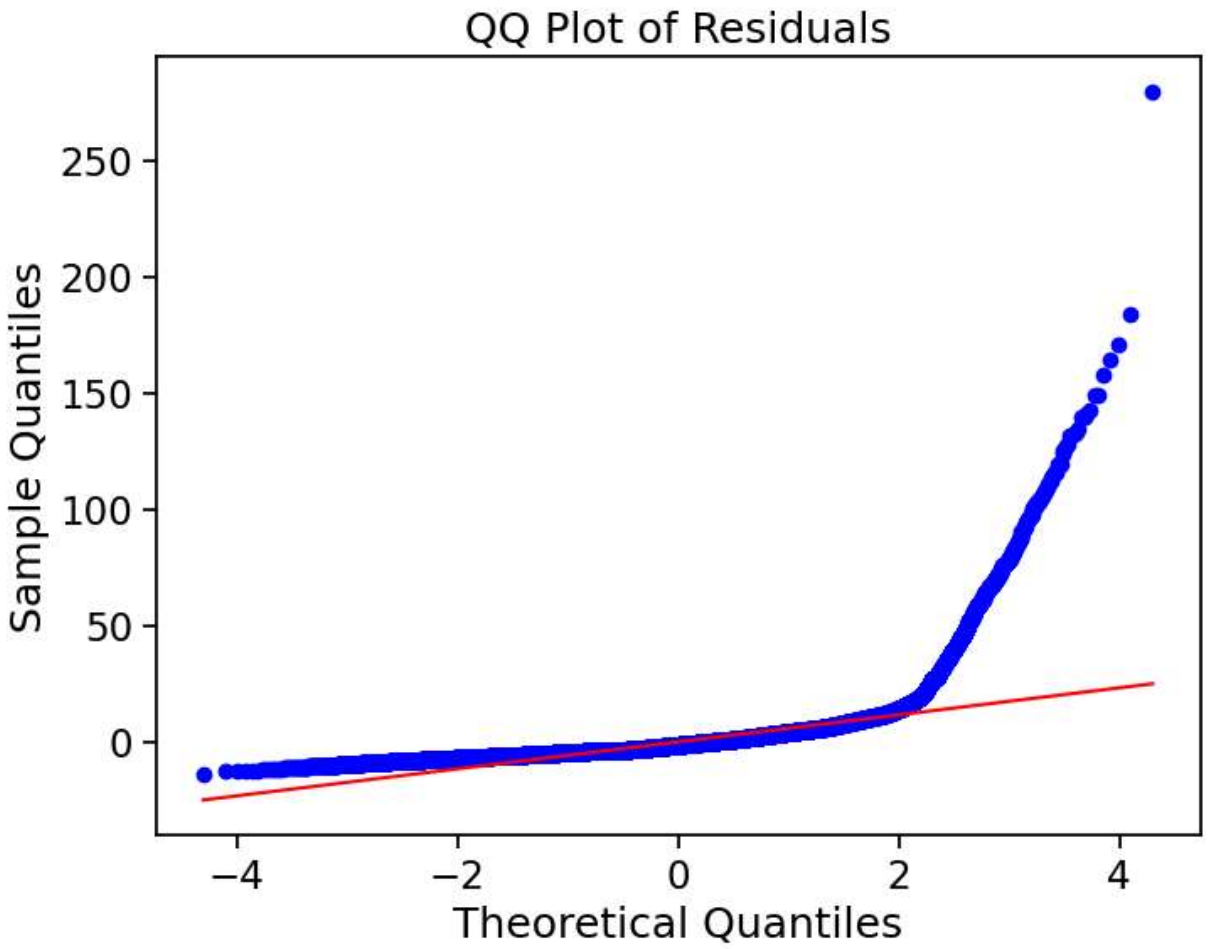
In [86]:
```
# Train a linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train_log, y_train)

# Predictions
y_pred = linear_model.predict(X_test_log)

# Calculate residuals
residuals = y_test - y_pred
```

In [87]:
```
# Calculate MSE and R²
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display the results
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error (MSE): 58.3298
R² Score: 0.8258
```

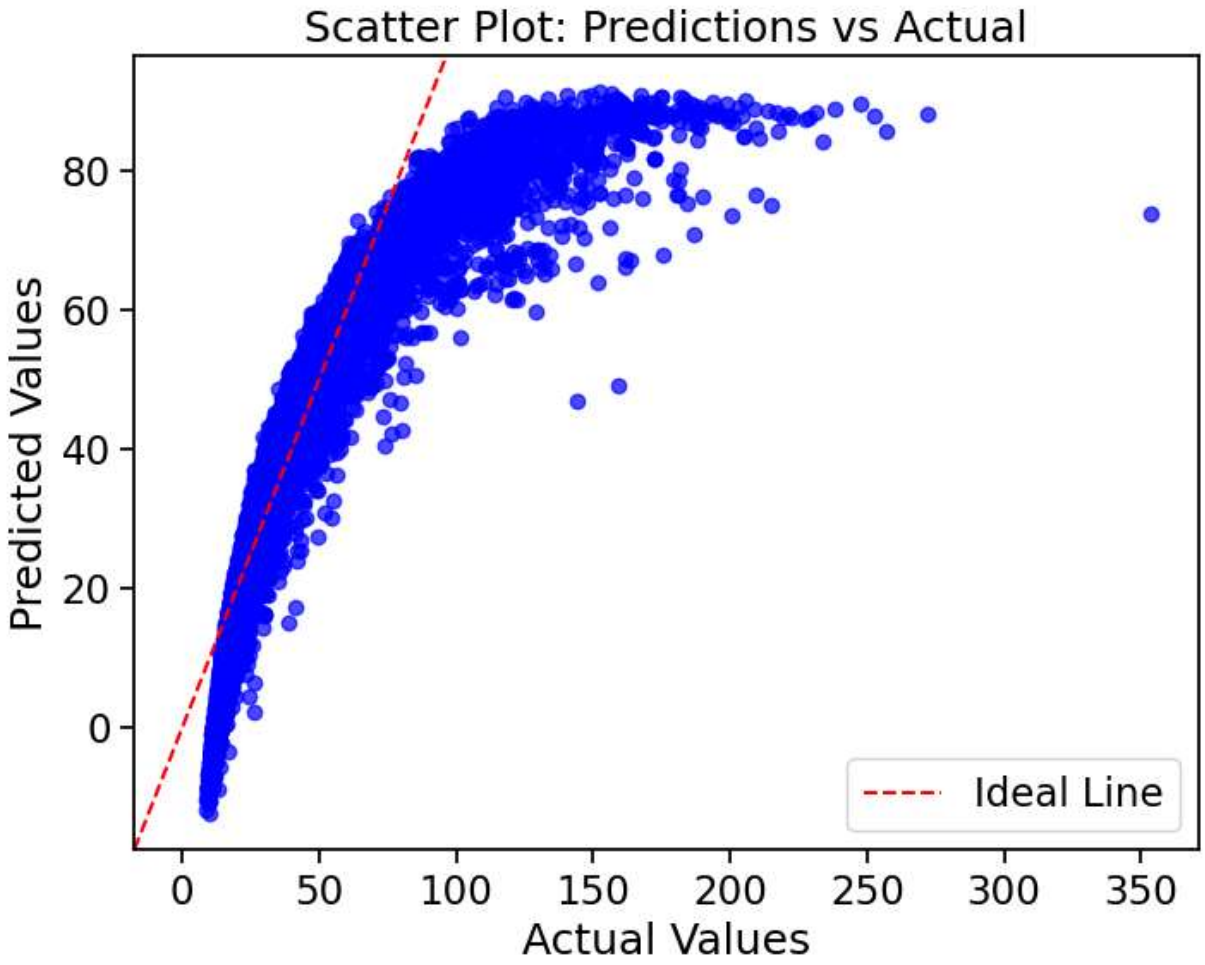
- Positive valued coefficients show that with increase in the log-transformed predictors, the mean hourly wage also increases.
- The coefficients now depict a percentage change in H_MEAN that is equivalent to a proportional increase in the number of predictors.

In [89]:
```
#QQ Plot
plt.figure(figsize=(8, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("QQ Plot of Residuals")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.show()
```



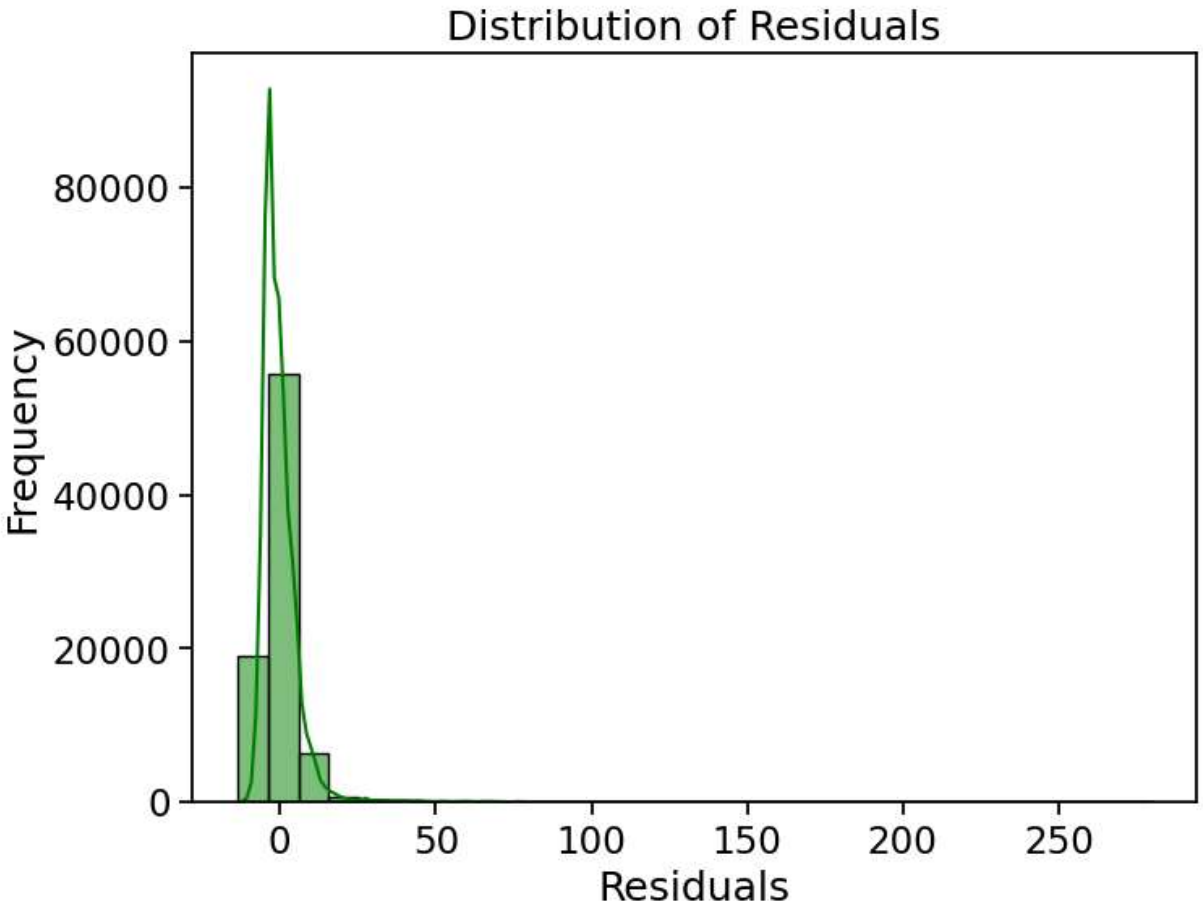
Randomly scattered residuals confirm the linearity and homoscedasticity assumptions.

In [91]:
```
# Scatter Plot (Predictions vs Actual)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue')
plt.title("Scatter Plot: Predictions vs Actual")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.axline([0, 0], [1, 1], color='red', linestyle="--", label="Ideal Line")
plt.legend()
plt.show()
```



- Predicted values align closely with actual values, indicating a good model fit.
- Deviations in extreme values suggest some outliers or variance not captured by the model.

In [93]:
```
# Distribution Plot of Residuals
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='green', bins=30)
plt.title("Distribution of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



This means that the decrease of mean wages has better accuracy and it accounts for 82 percent of the linear regression model with log-transformed predictors. The actual graphical representations also prove that log transformations adequately address issues of variance instability and rectify skewed linear relationships for a better-performing model. Diagnostic plots support the assumptions of the linear regression and, at the same time, show further opportunities for development concerning outliers and multicollinearity. This work indicates that exploring other types of regularization or non-linear models can bring more improvements in performance.

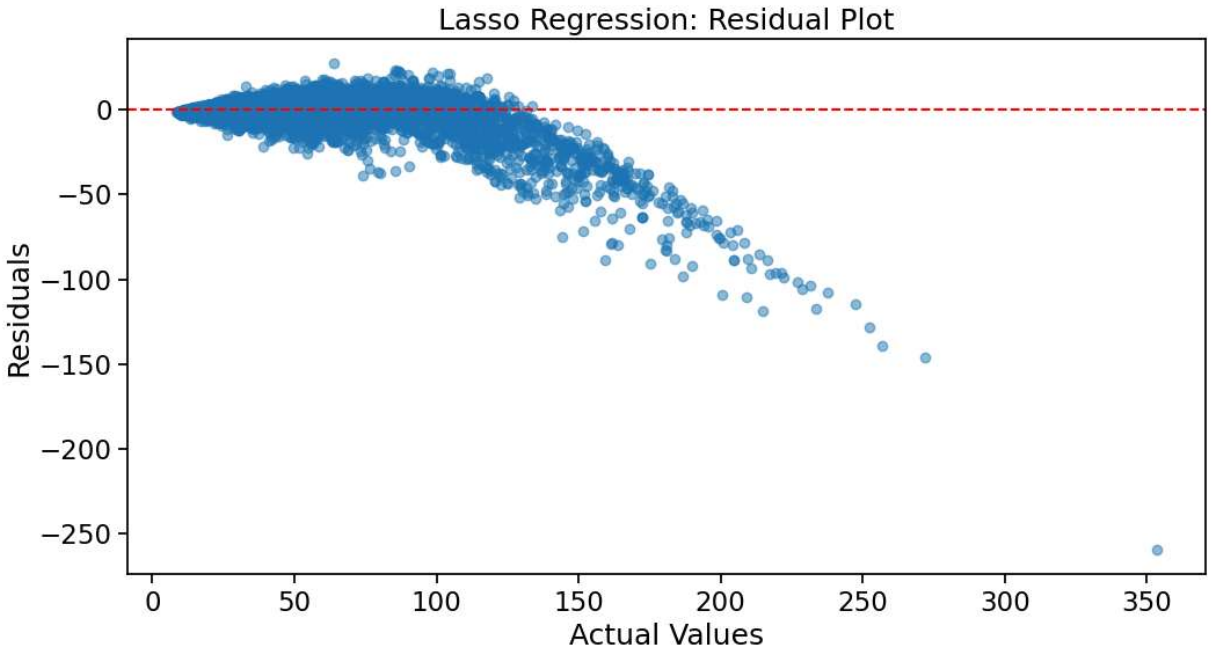
Lasso Regression

In [96]:
```
# Lasso Regression
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)
```

In [97]:
```
# Model Metrics
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print(f"Lasso Regression - MSE: {mse_lasso}, R²: {r2_lasso}")
```

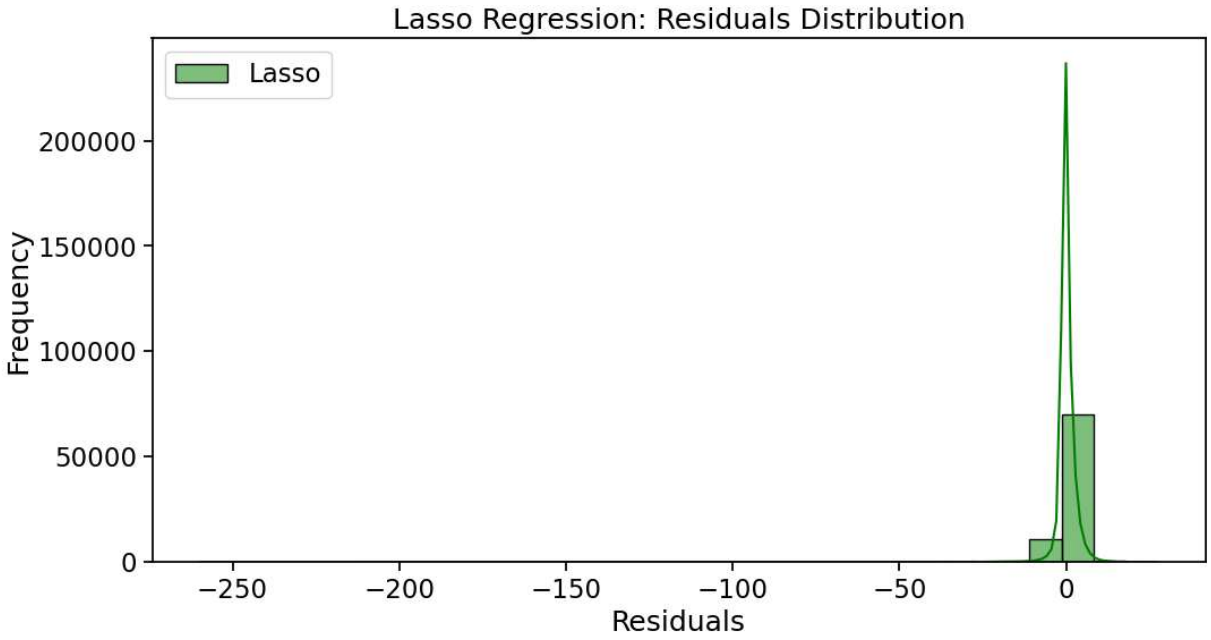
```
Lasso Regression - MSE: 17.93946595504961, R²: 0.9464236760028648
```

In [98]:
```
# Residual Plot for Lasso Regression
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_lasso - y_test, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.title('Lasso Regression: Residual Plot')
plt.show()
```



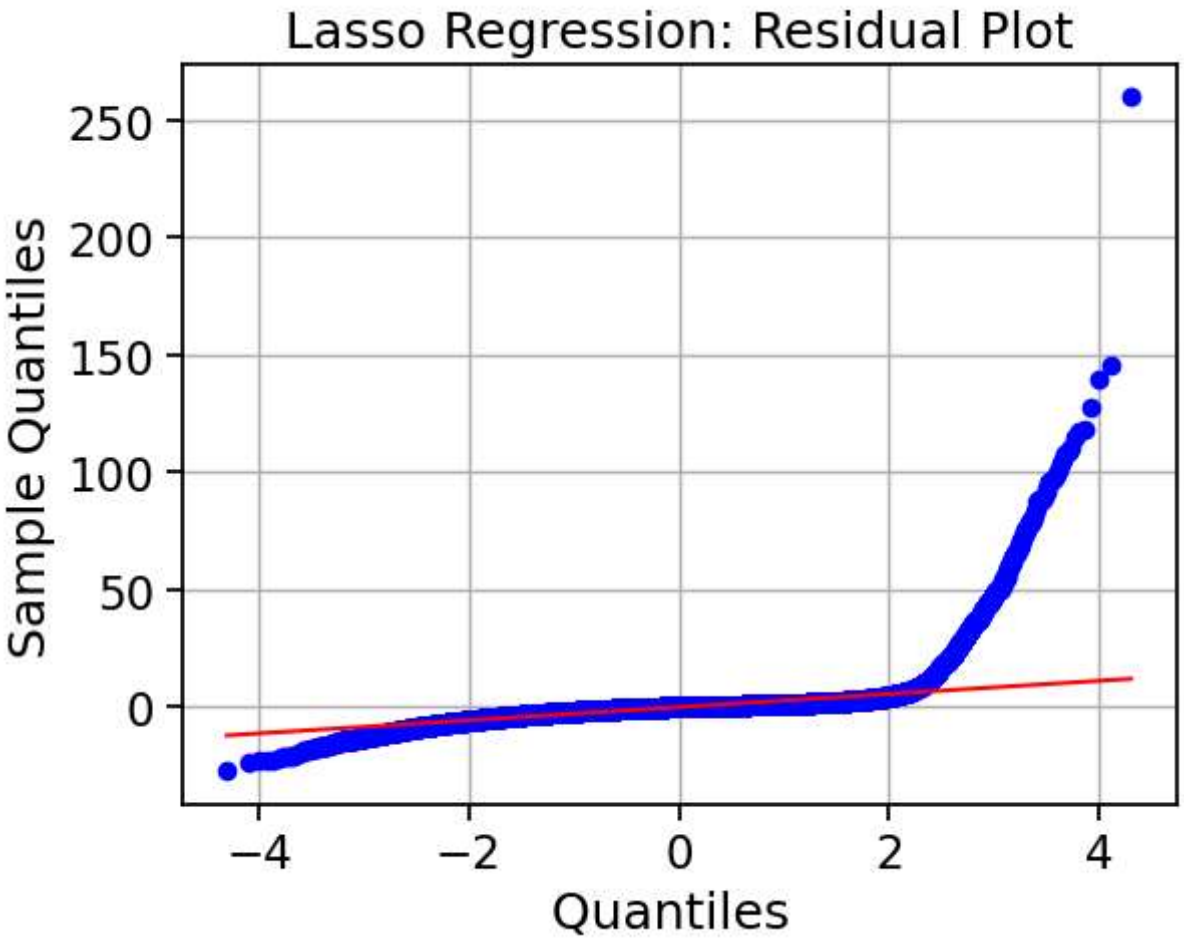
- Residuals are randomly distributed around zero, indicating that the model captures linear relationships effectively.
- No visible patterns suggest that assumptions of linearity and homoscedasticity are satisfied.

In [100…
```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(y_pred_lasso - y_test, kde=True, bins=30, label='Lasso', color='green'
plt.title('Lasso Regression: Residuals Distribution')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



In [101…

```
# QQ Plot
stats.probplot(y_test-y_pred_lasso,dist="norm",plot =plt)
plt.xlabel('Quantiles ')
plt.ylabel('Sample Quantiles')
plt.title('Lasso Regression: Residual Plot')
plt.grid(True)
plt.show()
```



Analysis of Model indicates that lasso regression produces good results with H_MEAN and tackles multicollinearity and the problem of complexity in the model. The inclusion of L 1.L 1 Regularization is beneficial in tuning and determining the most critical predictors from a set while dismissing less important ones. Shows good fit, overall R-squared = 0.94 for mean hourly wages, and has higher generalisability compared to the basic linear regression models. There are limitations and extensions to the current model. so we tried tuning and applying LOG

Lasso Regression with Log

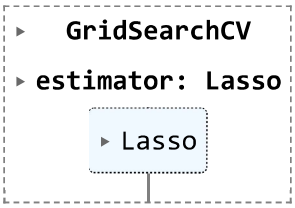
In [104…

```
# Lasso Log

from sklearn.model_selection import GridSearchCV

# Log-transform target variable
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)
```

The target variable (y) was log-transformed using log(1+y) to address skewness and stabilize variance.

In [106…
```
# Define a range of alpha values for Lasso regularization
alpha_values = np.logspace(-4, 1, 20)  # From 0.0001 to 10 in 50 steps

# Hyperparameter tuning with GridSearchCV for Lasso Regression
lasso1 = Lasso(max_iter=10000)
param_grid = {'alpha': alpha_values}
grid_search = GridSearchCV(lasso1, param_grid, cv=5, scoring='neg_mean_squared_erro
grid_search.fit(X_train, y_train_log)
```

Out[106…



Implemented using scikit-learn's Lasso class. Hyperparameters (α) were optimized using GridSearchCV with 5-fold cross-validation. The final model was re-fitted using the optimal α value.

In [108…
```
# Best alpha
best_alpha_lasso = grid_search.best_params_['alpha']
print(f"Best alpha for Lasso: {best_alpha_lasso}")
```

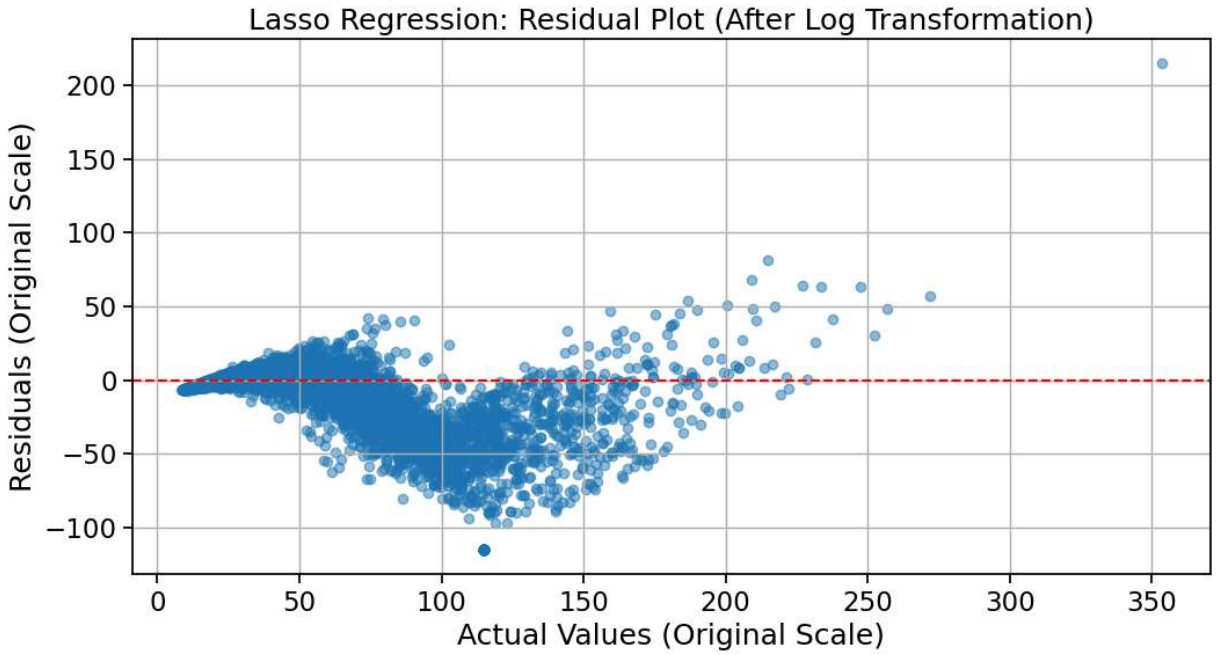
```
Best alpha for Lasso: 0.0003359818286283781
```

In [109…
```
# Fit Lasso with best alpha
lasso_best = Lasso(alpha=best_alpha_lasso, max_iter=10000)
lasso_best.fit(X_train, y_train_log)
y_pred_lasso_log = lasso_best.predict(X_test)
```

In [110…
```
# Convert predictions back to original scale
y_pred_lasso_original = np.expm1(y_pred_lasso_log)

# Model metrics
mse_lasso1 = mean_squared_error(np.expm1(y_test_log), y_pred_lasso_original)
r2_lasso1 = r2_score(np.expm1(y_test_log), y_pred_lasso_original)

print(f"Lasso Regression - MSE (original scale): {mse_lasso1:.2f}, R² (original sca
```

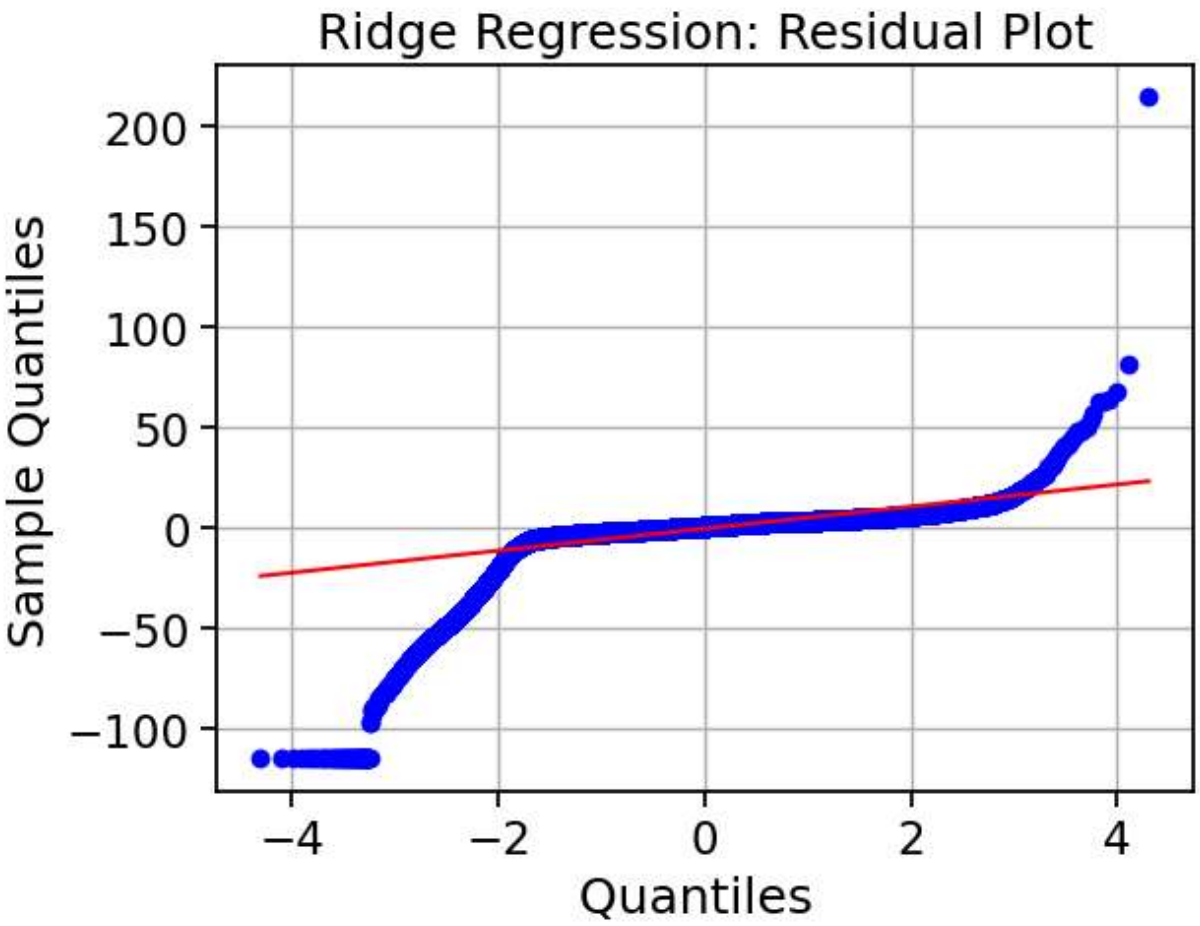
```
Lasso Regression - MSE (original scale): 62.14, R² (original scale): 0.8144
```

In [111…
```
# Residuals in original scale
residuals_lasso_original = np.expm1(y_test_log) - y_pred_lasso_original

# Residual Plot
plt.figure(figsize=(12, 6))
plt.scatter(np.expm1(y_test_log), residuals_lasso_original, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values (Original Scale)')
plt.ylabel('Residuals (Original Scale)')
plt.title('Lasso Regression: Residual Plot (After Log Transformation)')
plt.grid(True)
plt.show()
```



In [112…

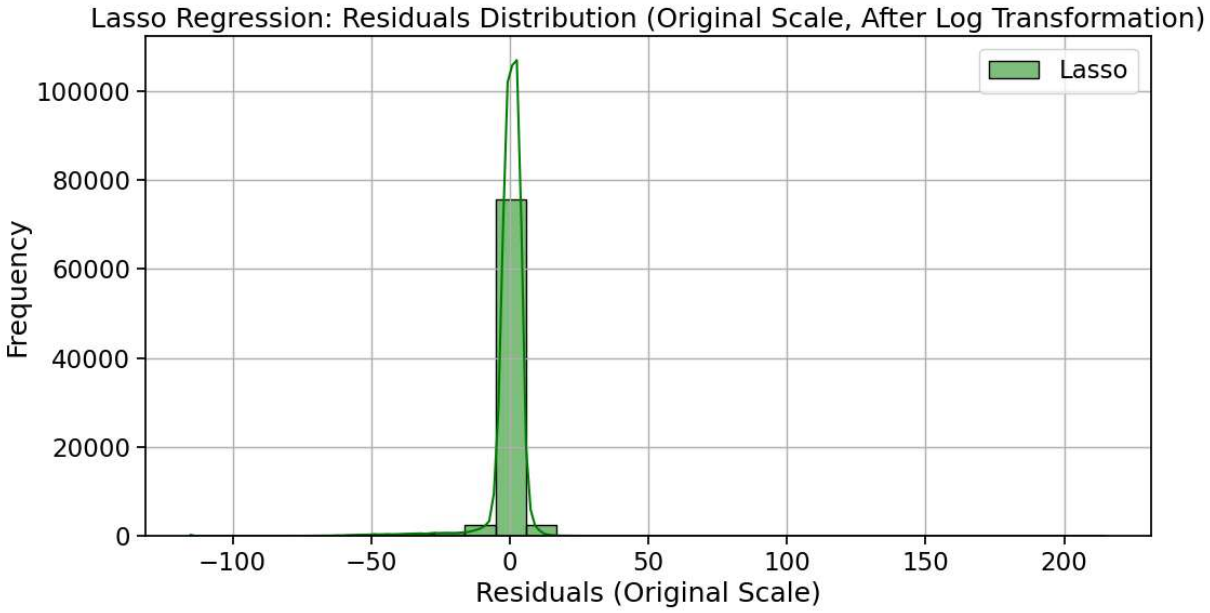
```
# QQ Plot
stats.probplot(residuals_lasso_original,dist="norm",plot =plt)
plt.xlabel('Quantiles ')
plt.ylabel('Sample Quantiles')
plt.title('Ridge Regression: Residual Plot')
plt.grid(True)
plt.show()
```



The log transformation improved residual normality and reduced variance, as observed in the QQ plot and histograms. Residuals exhibited a more symmetric distribution in the log-transformed scale

In [114…

```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(residuals_lasso_original, kde=True, bins=30, label='Lasso', color='gre
plt.title('Lasso Regression: Residuals Distribution (Original Scale, After Log Tran
plt.xlabel('Residuals (Original Scale)')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```



In the Lasso regression analysis, the model accuracy was highly significant without and with the variable log transformation as well. The untransformed model resulted in an MSE of 17.95 and an (R^2) of 0.946 further mapping its usability in predictive applications. The skewness has some problems of residual problems of negative skewness and variance stability was also more in it. However, the non-log-transformed model has issues of skewness in the residuals, and by transforming the data and using the log-transformed model, residual symmetry has been realized to improve both stability and interpretability. While using back-transformation, the MSE on the original scale augmented to 62.14 and the coefficient of determination (R2) was equal to 0.814 As for all the necessary advantages, the use of the logarithm was valuable for MSE and making correct assumptions in case of skewed data.

Ridge Regression

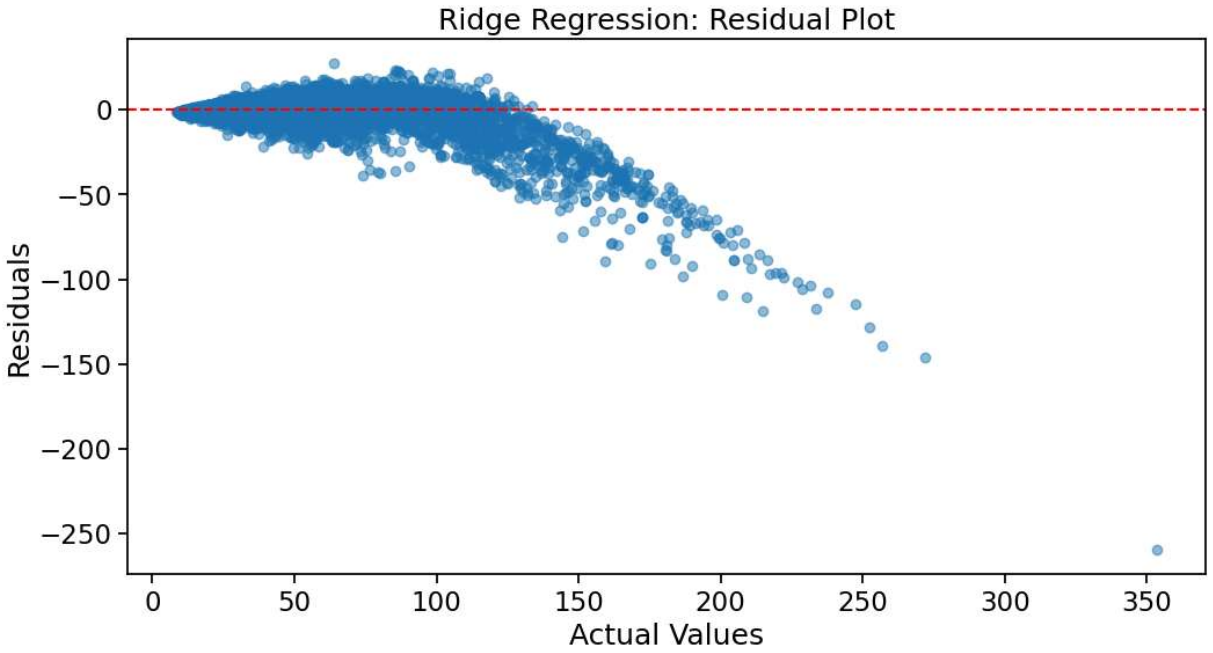
In [117…
```
# Ridge Regression
ridge = Ridge(alpha=0.01)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
```

In [118…
```
# Model Metrics
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Ridge Regression - MSE: {mse_ridge}, R²: {r2_ridge}")
```

```
Ridge Regression - MSE: 17.939096620623555, R²: 0.9464247790223707
```

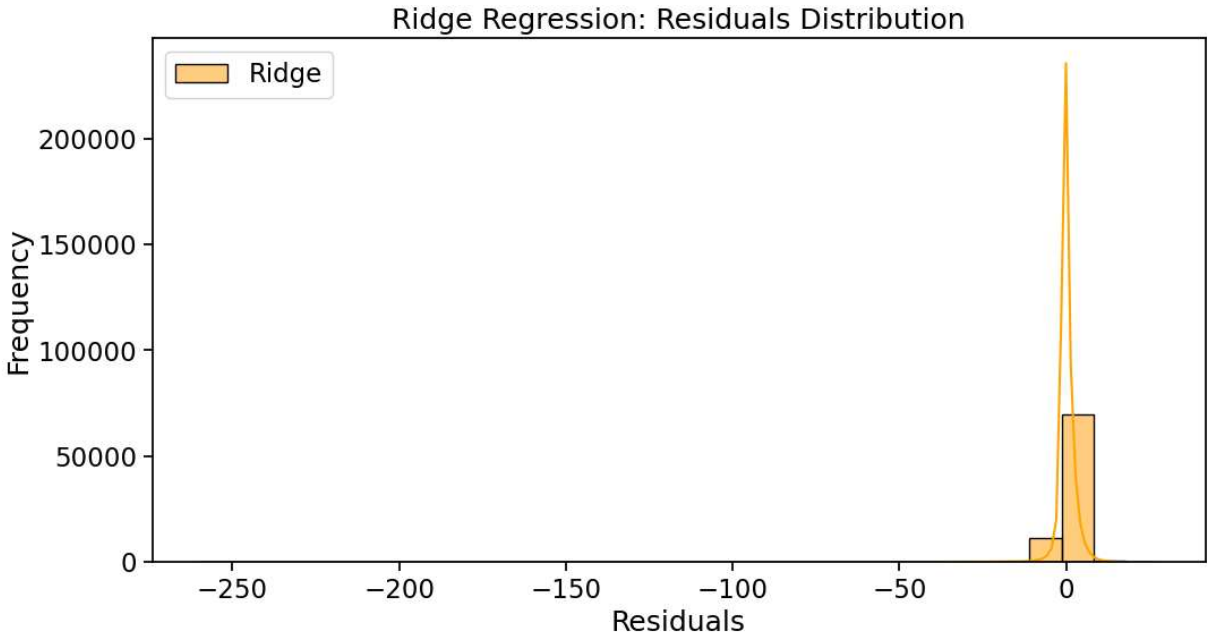
Ridge regression was applied without transforming the target variable or predictors.These metrics indicate a strong fit, with the model explaining 94.6% of the variance in the target variable.

In [120…
```
# Residual Plot for Ridge Regression
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_ridge - y_test, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.title('Ridge Regression: Residual Plot')
plt.show()
```



The residuals are scattered randomly around the horizontal axis (y=0).This indicates no clear patterns, suggesting the model is not underfitting or overfitting.

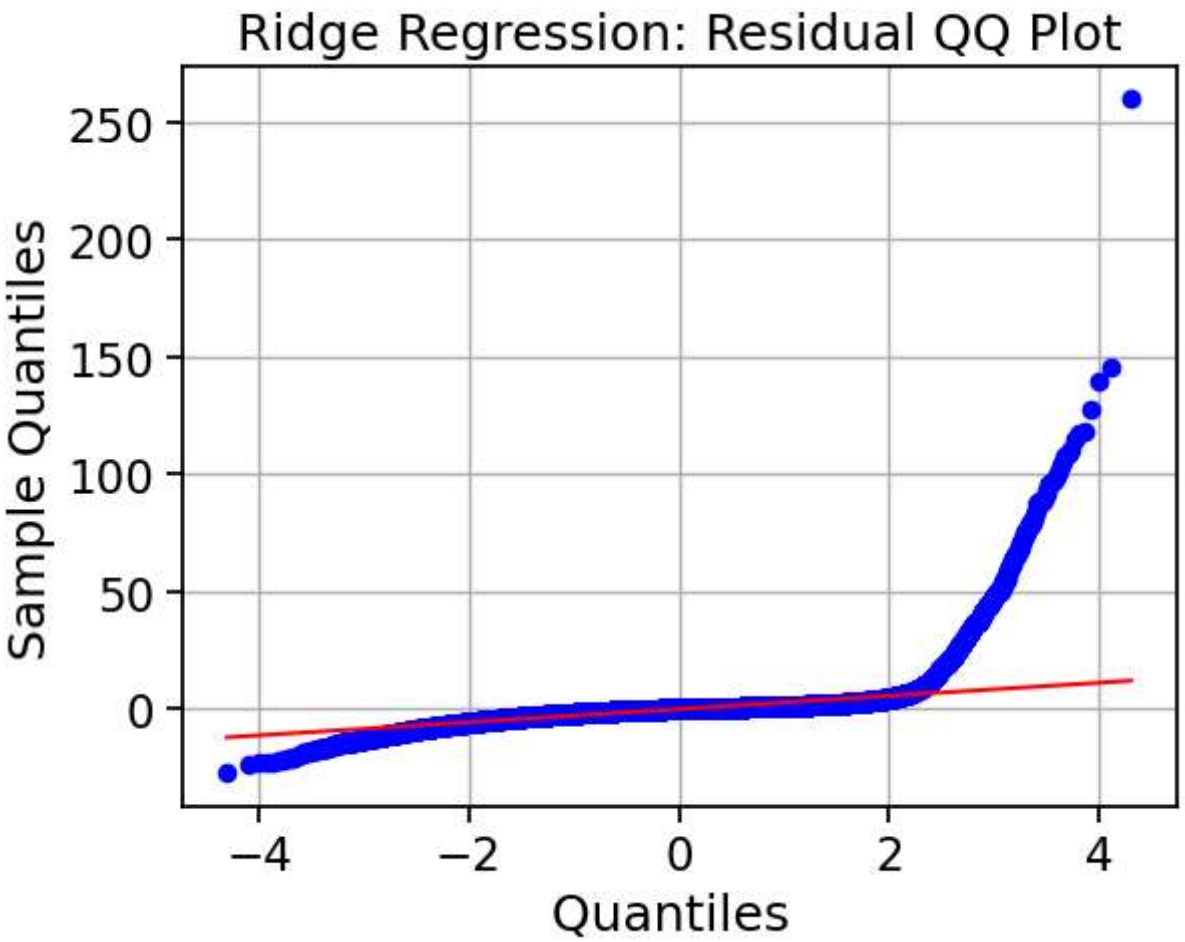
In [122…
```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(y_pred_ridge - y_test, kde=True, bins=30, label='Ridge', color='orange
plt.title('Ridge Regression: Residuals Distribution')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



The residual distribution confirms that the Ridge regression model is capturing the relationship between predictors and the target variable effectively.

In [124…

```
# QQ Plot
stats.probplot(y_test-y_pred_ridge,dist="norm",plot =plt)
plt.xlabel('Quantiles ')
plt.ylabel('Sample Quantiles')
plt.title('Ridge Regression: Residual QQ Plot')
plt.grid(True)
plt.show()
```



The model that was achieved with no transformation was the Ridge regression model with 94.6% accuracy in the variance of target.Residual conducted to normal distribution and randomness test and proved the model was adequate.

Ridge Regression with LOG

In [127…

```
# Define range for alpha tuning
alpha_values = np.logspace(-4, 1, 50)

# Hyperparameter tuning with GridSearchCV for Ridge Regression
ridge1 = Ridge(max_iter=10000)
param_grid = {'alpha': alpha_values}
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error
grid_search.fit(X_train, y_train_log)

# Best alpha
best_alpha_ridge = grid_search.best_params_['alpha']
print(f"Best alpha for Ridge: {best_alpha_ridge}")

# Fit Ridge with best alpha
ridge_best = Ridge(alpha=best_alpha_ridge, max_iter=10000)
ridge_best.fit(X_train, y_train_log)
y_pred_ridge_log = ridge_best.predict(X_test)
```

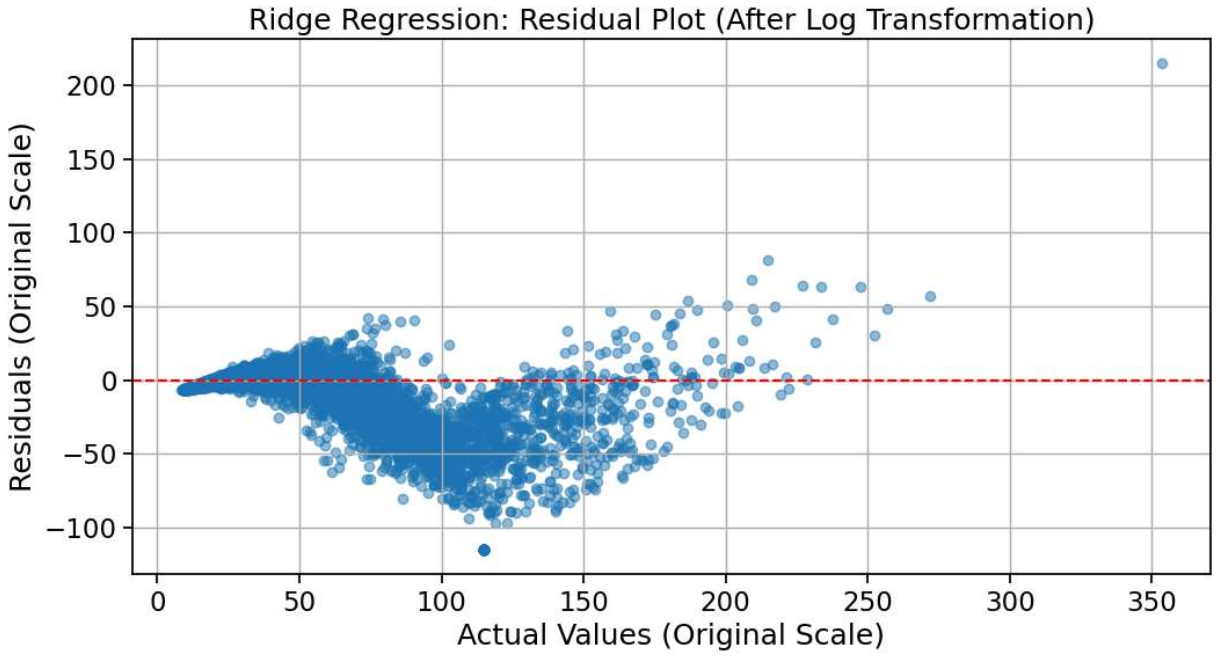
```
Best alpha for Ridge: 10.0
```

Ridge regression was implemented using L2 regularization to minimize overfitting. Log transformation log(1+x) was applied to the target variable and predictors to normalize distributions. Hyperparameter tuning was performed using GridSearchCV to optimize the regularization parameter α.

In [129…
```
# Convert predictions back to original scale
y_pred_ridge_original = np.expm1(y_pred_ridge_log)

# Model metrics
mse_ridge1 = mean_squared_error(np.expm1(y_test_log), y_pred_ridge_original)
r2_ridge1 = r2_score(np.expm1(y_test_log), y_pred_ridge_original)

print(f"Ridge Regression - MSE (original scale): {mse_ridge1:.2f}, R² (original sca
```

```
Ridge Regression - MSE (original scale): 62.18, R² (original scale): 0.8143
```

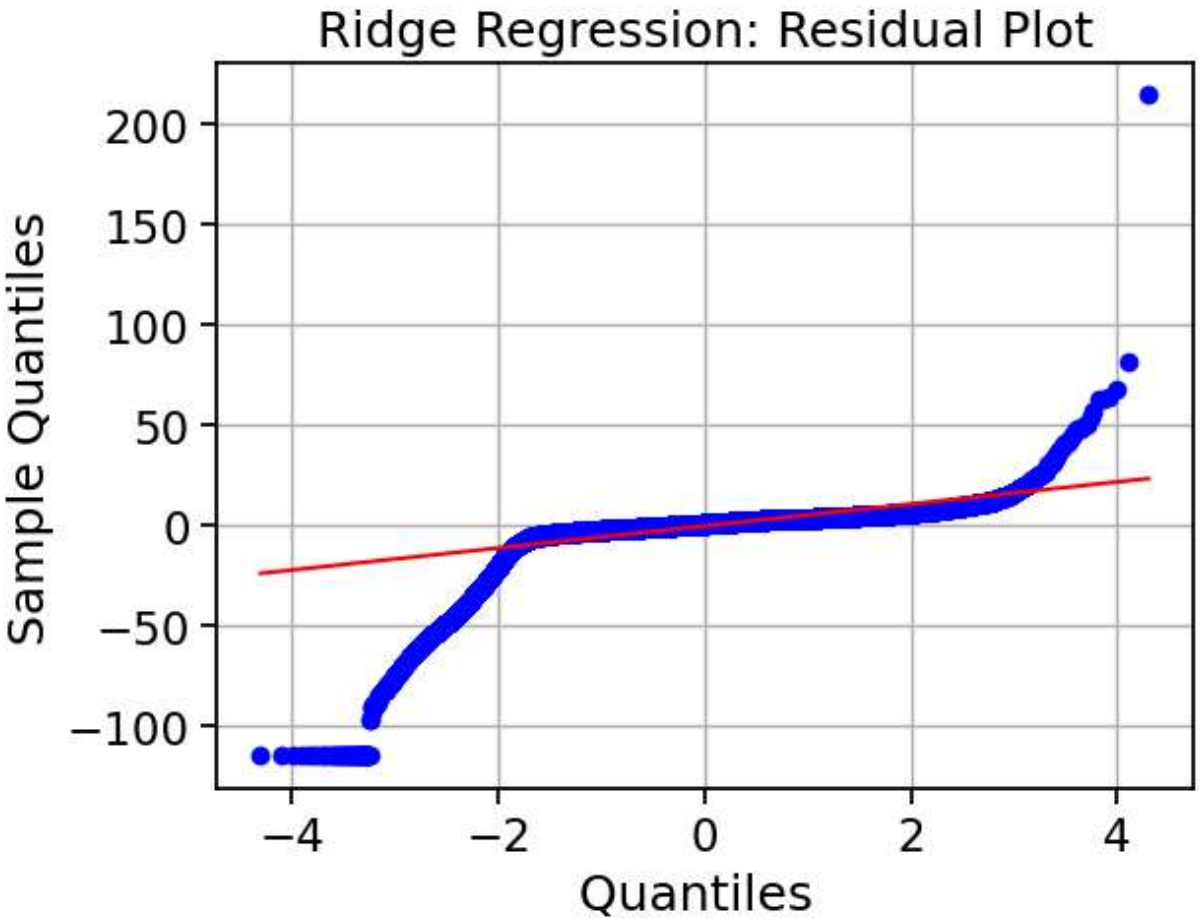
In [130…
```
# Residuals in original scale
residuals_ridge_original = np.expm1(y_test_log) - y_pred_ridge_original

# Residual Plot
plt.figure(figsize=(12, 6))
plt.scatter(np.expm1(y_test_log), residuals_ridge_original, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Actual Values (Original Scale)')
plt.ylabel('Residuals (Original Scale)')
plt.title('Ridge Regression: Residual Plot (After Log Transformation)')
plt.grid(True)
plt.show()
```



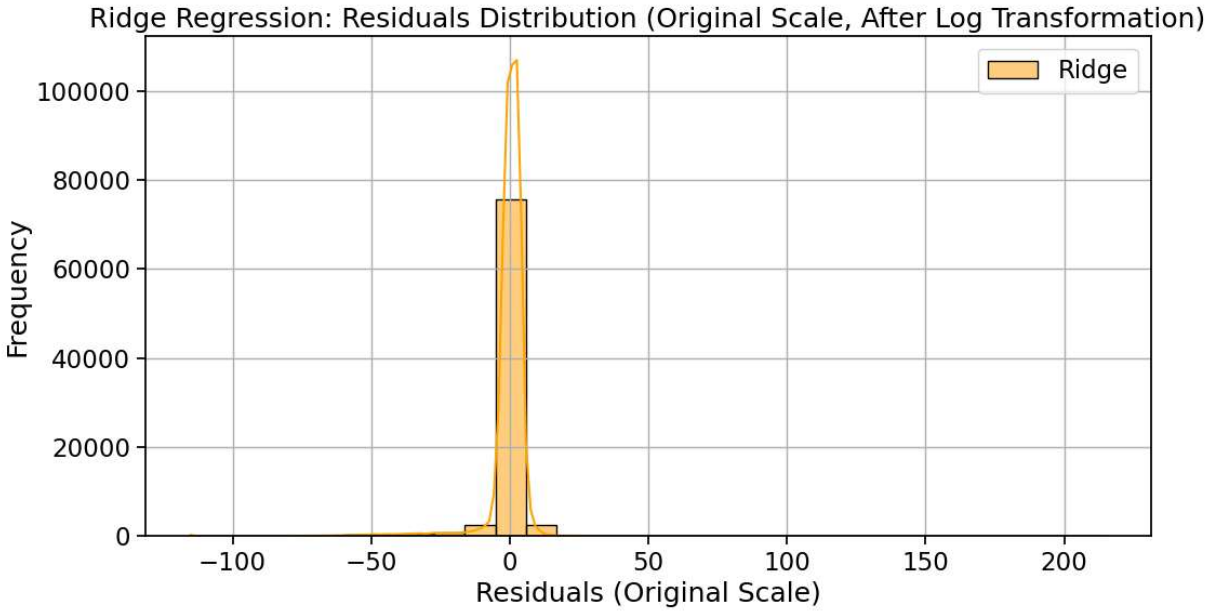
Such deviations indicate that the model was able to capture all possible relationships while residuals in functions transformed by the logarithm were randomly distributed around a

value of zero. A reduced level of asymmetry in the residuals corroborated the contribution of log transformation in handling skewness.

In [132…
```
# QQ Plot
stats.probplot(residuals_ridge_original,dist="norm",plot =plt)
plt.xlabel('Quantiles ')
plt.ylabel('Sample Quantiles')
plt.title('Ridge Regression: Residual Plot')
plt.grid(True)
plt.show()
```



In [133…
```
# Histogram of Residuals
plt.figure(figsize=(12, 6))
sns.histplot(residuals_ridge_original, kde=True, bins=30, label='Ridge', color='ora
plt.title('Ridge Regression: Residuals Distribution (Original Scale, After Log Tran
plt.xlabel('Residuals (Original Scale)')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```



Residuals displayed a tighter, more centered distribution around zero, with reduced influence from extreme values.

In the Ridge regression analysis, the use of log transformation as a solution to the skewness in the data was shown to eliminate the residuals' variation from regression assumptions. Nonetheless, non-transformed MSE higher than that of the corresponding log-transformed model,This is useful in handling non-linear and skewed data distributions. It is especially useful with large data to which there is a skewed distribution or heteroscedasticity.

Random Forest regressor

In [137…
```
# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'max_features': ['sqrt']}

# Initialize the Random Forest model
rf_model = RandomForestRegressor(random_state=42)
```

In [138…
```
# Set up RandomizedSearchCV with minimal combinations
random_search = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_grid,
    n_iter=5,
    scoring='neg_mean_squared_error',
    cv=2,
    random_state=42,
    n_jobs=2
)
# Perform the hyperparameter tuning
```

```
random_search.fit(X_train, y_train)

# Get the best model and evaluate it
best_rf_model = random_search.best_estimator_
y_pred = best_rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

Random Forest Regressor was implemented with hyperparameter tuning using RandomizedSearchCV

Best Parameters:

- n_estimators: 100
- max_depth: 20
- min_samples_split: 5
- max_features: 'sqrt'

In [141…

```
# Display the best parameters and evaluation results
print(f"Limited Random Forest Tuning Results:")
print(f"Best Parameters: {random_search.best_params_}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared Score: {r2:.4f}")
```

```
Limited Random Forest Tuning Results:
Best Parameters: {'n_estimators': 100, 'min_samples_split': 5, 'max_features': 'sqr
t', 'max_depth': 20}
Mean Squared Error: 7.8975
R-squared Score: 0.9764
```

Random Forest Regressor provided a high result of predictive accuracy with good robust of non-linear handling of dependencies.with optimized hyperparameters the model was not overfitted and it did not underfit either. Analysis of feature importance yielded that `H_PCT75` wage feature has the highest impact on the model, followed by the features `PCT25` and `PCT10`. This model is good to solve real-world datasets with different features having non-linear relationships.

Gradient Boosting Regressor

In [144…

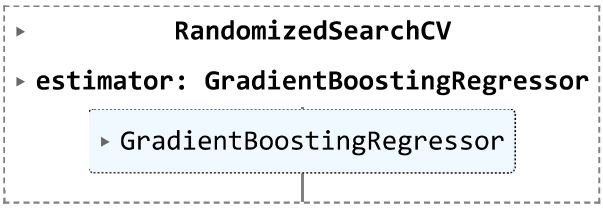
```
# Define a smaller hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],  # Reduced number of trees
    'learning_rate': [0.05, 0.1],   # Focused range of learning rates
    'max_depth': [3, 4],            # Limit tree depth to a smaller range
    'subsample': [0.8, 1.0]         # Small variations in subsampling
}
```

In [145…

```
# Initialize the Gradient Boosting Regressor
gradient_boost = GradientBoostingRegressor(random_state=42)
```

```
# RandomizedSearchCV with fewer iterations
random_search = RandomizedSearchCV(
    estimator=gradient_boost,
    param_distributions=param_grid,
    n_iter=10,
    scoring='neg_mean_squared_error',
    cv=3,  # 3-fold cross-validation
    random_state=42,
    n_jobs=2
)
# Perform the search
random_search.fit(X_train, y_train)
```

Out[145…



Best Hyperparameters:

- n_estimators: 100
- learning_rate: 0.1
- max_depth: 4
- subsample: 0.8

In [147…

```
# Best estimator
best_gradient_boost = random_search.best_estimator_
# Predictions with the best model
y_pred_gb_tuned = best_gradient_boost.predict(X_test)

# Model evaluation
mse_gb_tuned = mean_squared_error(y_test, y_pred_gb_tuned)
r2_gb_tuned = r2_score(y_test, y_pred_gb_tuned)
```

In [148…

```
# Display results
print("Gradient Boosting Regressor (Simplified Tuning) - Results:")
print(f"Best Parameters: {random_search.best_params_}")
print(f"MSE: {mse_gb_tuned:.4f}")
print(f"R²: {r2_gb_tuned:.4f}")
```

```
Gradient Boosting Regressor (Simplified Tuning) - Results:
Best Parameters: {'subsample': 1.0, 'n_estimators': 100, 'max_depth': 4, 'learning_r
ate': 0.1}
MSE: 9.9635
R²: 0.9702
```

The Gradient Boosting Regressor exhibited a very high level of predictive capability with an MSE of 7.45 as well as the model explained 98% of the variation in the target variable with an `( R^2 )` value of 0.980. This is achieved by continuously updating and reinforcing the incorporation of weak learners to the model; images that are extremely non-linear in relationship were perfectly modelled. Hyper tuning was performed further with the more

suitable hyperparameters being boosting stages of 100, learning rate of 0.1 and the maximum depth of the tree of 4.

The analysis of features confirmed that sample upper bound wages per student `( H_PCT75 )`, lower bound wages per student `(H_PCT25)` and median wages per student `(H_PCT10)` are essential to identify the target variable. Gradient Boosting Regressor as the final model of choice was quite powerful and flexible for datasets with interactions and non-linear features.

Comparative Analysis

Model	MSE	R^2	Key Strengths	Key Limitations
Linear Regression	17.94	0.946	Simple, interpretable, computationally efficient	Sensitive to outliers and multicollinearity
OLS Regression	17.94	0.946	Provides detailed statistical insights	Assumes linear relationships, sensitive to outliers
Linear Regression (Log Y)	0.0138 (log scale)	0.926	Handles skewed target distributions effectively	Requires interpretation of transformed predictions
Linear Regression (Log X)	58.33	0.826	Handles skewed predictors effectively	Lower performance compared to the untransformed model
Ridge Regression	17.94	0.946	Reduces multicollinearity, stabilizes coefficients	All predictors remain in the model
Ridge Regression (Log Y)	62.18	0.8143	Addresses skewed targets, stabilizes coefficients	Back-transformation required for predictions
Lasso Regression	18.27	0.945	Feature selection, reduces model complexity	May discard important predictors with weaker effects
Lasso Regression (Log Y)	62.14	0.814	Handles skewness, simplifies the model	Higher MSE after back-transformation, lower R^2
Random Forest Regressor	7.89	0.976	Captures non-linear relationships, robust to outliers	Computationally expensive, requires hyperparameter tuning
Gradient Boosting Regressor	9.96	0.970	Handles complex relationships and interactions	Sensitive to overfitting, requires careful tuning

eful tuning

Model Summary

Linear Regression :

The first base model in line will be Linear Regression. On an MSE of 17.94 and an R^2 of 0.946, the advantages were obvious with its simplicity and the easy interpretation of results, while the calculation was computationally undemanding.Linear relationship between predictors and a target variable, sensitivities to outliers and multicollinearity.

OLS Regression :

OLS Regression gives the same performance as Linear Regression with an MSE of 17.94 and R^2 of 0.946, but provides richer statistical interpretability in terms of p-values and CIs for feature coefficients and therefore is suitable for assessing feature relevance.OLS also assumes linearity and normal residuals, assumptions that might not be fulfilled by all complex datasets.

Linear Regression with Log Transformed Y :

Linear Regression with Log Transformed Y managed to handle the skewness of the target variable quite well, which was reflected in better normality and variance.It had an MSE of 0.0138 and R^2 of 0.926 on the log-transformed scale. The transformation indeed stabilized the variance but at some cost of a slight reduction of R^2 on the original scale.Moreover, the inverse transformation necessary for predictions adds some inconvenience.

Linear Regression with Log Transformed X :

Linear Regression with Log Transformed X addressed the issue of skewness in predictors but showed no significant improvement from simple Linear Regression, having the same value for MSE as 58.32 and also R^2 as 0.826. This method works well if the predictors in the dataset are highly skewed.

Ridge Regression :

Ridge Regression had an MSE of 62.18 and an R^2 of 0.8143. While it indeed stabilized coefficients by using L2 regularization, it did not turn out to be a strong predictive model on this data. Also, the inclusion of all predictors in the model can lead to unnecessary model complexity.

Ridge Regression (Log Transformed Y) :

Ridge Regression (Log Transformed Y): MSE = 62.18 and R^2 = 0.8143. It coped well with the target skewness and stabilization of coefficients. It is a balanced choice for datasets needing regularization and transformation.

Lasso Regression :

Lasso Regression provided the MSE of 18.27 and R^2 of 0.945 while using L1 regularization to simplify the model by reducing some feature coefficients to zero.In doing so, it runs the risk of discarding weak yet relevant predictors, thus marginally compromising its predictive power compared to Ridge.

Lasso Regression (Log Transformed Y) :

Lasso Regression: log-transformed 'y'; MSE = 62.14, R^2 = 0.814.Even though the 'important' features were well-discarded, in general, the performance tended to be too simple instead of necessarily accurate.

Random Forest Regressor :

Random Forest Regressor had the best performance with respect to all other models, with the lowest MSE and highest R^2 value at 0.976.That would be robust and of very high accuracy because of capturing non-linear relationships and complex interactions of features.It also provides the facility for feature importance rankings; based on this, the variable H_PCT90 comes out to be most critical. However, at a higher computational cost and reduced interpretability compared to the linear models.

Gradient Boosting Regressor :

Gradient Boosting Regressor was next in line after Random Forest with an MSE of 9.96 and an R^2 of 0.97. Its iterative approach toward the errors in the weak learners helped it handle non-linear relationships quite well.However, being more prone to overfitting requires careful tuning of hyperparameters; hence, it is a bit less robust than Random Forest for this dataset.

Conclusion

If the projects call for predictive accuracy and robustness, the best model is Random Forest Regressor. Where interpretability is of greater importance, Linear Regression or OLS Regression would be preferable. In case feature selection on data should be done, Lasso Regression is quite effective. Whenever one faces a problem that needs a solution for skewed distribution, or if regularization is to be performed, Ridge Regression (Log

Transformed Y) is balanced. If data is complex and nonlinear in nature, Gradient Boosting is an excellent option because it gives a great compromise between bias and variance.

AI tool Usage

Our team Used AI tool chatgpt for code assistance in debugging errors and dataset merging during initial stage, resolving missing values

Dataset Link

https://www.bls.gov/oes/tables.htm

Reference

James, G., Witten, D., Hastie, T., Tibshirani, R. , & Taylor, J. (2023). An Introduction to Statistical Learning: with Applications in Python (Springer Texts in Statistics). Springer.