Predicting Salaries Data

This notebook is building a simple multiregression model to make predictions on salary. It uses salaries data that was cleaned in the Salaries_Preparation notebook.
There are 4 main goals:

- *Build a Baseline multiregression model that can be validated with Glassdoor data.*
- *Create an Expanded model, that contains parameters not available from Glassdoor.*
- *Extract the coefficient that will yield the effect of these parameters which are normalized properly with all the other variables.*
- *Make a personalized salary prediction.*

Table of Content:

In [21]:
```
import sys
sys.path.append('../')
from scripts.tableofcontent_generator import generate_toc
notebook_path = '../notebooks/Salaries_Prediction.ipynb'
```

In [22]:
```
toc_content = generate_toc(notebook_path)
print(toc_content)
```

Import libraries & data, general settings

In [15]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as dt
from datetime import datetime
import re
from IPython.display import HTML, display
```

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp as mc
from scipy.stats import levene, shapiro

# If the notebook is opened from the "notebooks" folder, we need to append the main directory to the "python path" so it sees all subfolders.
import sys
sys.path.append('../')
```

In [16]:
```
df_it = pd.read_csv('../data/cleaned/df_it.csv', low_memory=False)
df_k = pd.read_csv('../data/cleaned/df_k.csv', low_memory=False)
df_ai = pd.read_csv('../data/cleaned/df_ai.csv', low_memory=False)
```

Styles

In [17]:
```
# General Display settings

# Column display is supressed by default
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

#changing the display format
pd.set_option('display.float_format', lambda x: '%.2f' % x)

# Plotting format
#print(plt.style.available)
plt.style.use('seaborn-v0_8-whitegrid')
```

In [18]:
```
style_light_theme = """
<style>
h1 {
    background-color: #0e2e3b;
    color: white;
    font-size: 40px !important;
    font-weight: 700 !important;
    padding: 10px;
}
h2 {
    background-color: #07447E;
    color: white;
    font-size: 35px !important;
    font-weight: 700 !important;
    padding: 10px;
}
h3 {
    background-color: #047c98;
    color: white;
    font-size: 30px !important;
    font-weight: 700 !important;
    padding: 10px;
}
h4 {
    background-color: #0AB89E;
    color: white;
    font-size: 25px !important;
    font-weight: 700 !important;
    padding: 5px;
}
/* ---------------------------------  Conclusion class */
.c {
    background-color: #f7fe9a;
    color: black;
    padding: 10px 10px 10px 20px;        /* Top, Right, Bottom, Left */
    font-size: 16px;
    font-style: italic;
}
/* ---------------------------------  Note class */
.note {
    background-color: #f4fcc0;
    color: black;
    padding: 2px 10px 2px 20px;        /* Top, Right, Bottom, Left */
    font-size: 14px;
    font-style: italic;
```

```
}
</style>
"""
```

In [19]:
```
#display(HTML(style_dark_theme))
display(HTML(style_light_theme))
```

Preparing Dataframes

Western & Developed countries

In [11]:
```
western_countries = [
    'al', 'ad', 'am', 'at', 'az', 'by', 'be', 'ba', 'bg', 'hr',
    'cy', 'cz', 'dk', 'ee', 'fi', 'fr', 'ge', 'de', 'gr', 'hu',
    'is', 'ie', 'it', 'kz', 'xk', 'lv', 'li', 'lt', 'lu', 'mt',
    'md', 'mc', 'me', 'nl', 'mk', 'no', 'pl', 'pt', 'ro', 'ru',
    'sm', 'rs', 'sk', 'si', 'es', 'se', 'ch', 'tr', 'ua', 'gb',
    'va', 'ca', 'au', 'us'
]
```

In [12]:
```
western_countries = ['de', 'gb', 'nl', 'se', 'dk', 'be', 'fi', 'at', 'ch', 'ie', 'ca', 'au', 'us']
```

In [13]:
```
df_ai_w = df_ai.copy()
df_ai_w['country'] = df_ai_w['country'][df_ai_w['country'].isin(western_countries)]

df_k_w = df_k.copy()
df_k_w['country'] = df_k_w['country'][df_k_w['country'].isin(western_countries)]
```

In [14]:
```
# df_ai_d = df_ai.copy()
# df_ai_d['country'] = df_ai_d['country'][df_ai_d['country'].isin(developed_countries)]
#
# df_k_d = df_k.copy()
# df_k_d['country'] = df_k_d['country'][df_k_d['country'].isin(developed_countries)]
```

In [15]:
```
# GDP per capita data (in USD)
gdp_per_capita = {
    'fr': 40886, 'in': 2016, 'au': 65099, 'us': 76329, 'nl': 57025,
    'de': 48717, 'ie': 87947, 'ru': 15270, 'gr': 19829, 'ua': 4533,
    'pk': 1491, 'jp': 40066, 'br': 8697, 'kr': 31961,
    'by': 7888, 'ng': 2229, 'gb': 46125, 'se': 53755, 'mx': 10657,
    'ca': 54917, 'pt': 23758, 'pl': 17939, 'id': 4289, 'it': 34776,
    'cz': 23906, 'es': 31688, 'cl': 14938, 'hk': 46544, 'za': 6001,
    'ar': 10461, 'tr': 10674, 'il': 44162, 'tw': 34166, 'eg': 3801,
    'ma': 3585, 'hu': 18390, 'co': 6214, 'no': 89111, 'th': 7775,
    'ch': 93259, 'vn': 3704, 'sg': 59806, 'bd': 1964, 'ir': 2273,
    'pe': 7002, 'ke': 2066, 'ro': 15786, 'cn': 12710, 'be': 50126,
    'at': 52084, 'dz': 4094, 'nz': 45380, 'tn': 3840, 'ph': 3593,
    'my': 11109, 'dk': 61612, 'sa': 20619, 'ae': 43103, 'np': 1192,
    'lk': 3841, 'gh': 2396, 'et': 936, 'iq': 4922, 'ec': 6245,
    'kz': 10153, 'ug': 817, 'cm': 1500, 'zw': 1098,
    'lv': 21779, 'ge': 4804, 'lt': 25064, 'fi': 53012, 'hr': 15647, 'om': 25056, 'ba': 7568, 'ee': 28247,
    'mt': 34127, 'lb': 4136, 'si': 28439, 'mu': 10256, 'am': 7018, 'qa': 87661, 'ad': 41992, 'md': 5714,
    'uz': 2255, 'cf': 427, 'kw': 41079, 'cy': 32048, 'as': 19673, 'cr': 13365, 'pr': 35208, 'bo': 3600,
    'do': 8793, 'hn': 2736, 'bg': 12623, 'je': 55820, 'rs': 9260,  'lu': 125006
}
```

Glassdoor salary values

In [17]:
```
glassdoor_data = pd.DataFrame({
    'job_category': ['Data Analyst', 'Data Analyst', 'Data Engineer', 'Data Engineer', 'Software Engineer'],
    'seniority_level': ['junior', 'senior', 'senior', 'senior', 'senior'],
    'country': ['us', 'de', 'us', 'us', 'de'],
    'glassdoor_lower': [50000, 73000, 60000, 80000, 65000],
```

```
    'glassdoor_upper': [70000, 108000, 80000, 100000, 90000]
})
```

In [18]:
```
# Define the column names
columns = ['job_category', 'seniority_level', 'country', 'glassdoor_lower', 'glassdoor_upper']

# Create a list of tuples
glassdoor_data_rows = [
    ('Data Analyst', 'junior', 'us', 81000, 134000),
    ('Data Analyst', 'medior', 'us', 86000, 144000),
    ('Data Analyst', 'senior', 'us', 88000, 150000),
    ('Data Analyst', 'junior', 'de', 55000, 71000),
    ('Data Analyst', 'medior', 'de', 58000, 82000),
    ('Data Analyst', 'senior', 'de', 68000, 93000),
    ('Data Engineer', 'junior', 'us', 96000, 157000),
    ('Data Engineer', 'medior', 'us', 109000, 179000),
    ('Data Engineer', 'senior', 'us', 121000, 198000),
    ('Data Engineer', 'junior', 'de', 62500, 81000),
    ('Data Engineer', 'medior', 'de', 69000, 90100),
    ('Data Engineer', 'senior', 'de', 73500, 94000),
    ('Data Scientist/ ML Engineer', 'junior', 'us', 129000, 210000),
    ('Data Scientist/ ML Engineer', 'medior', 'us', 140000, 232000),
    ('Data Scientist/ ML Engineer', 'senior', 'us', 149000, 251000),
    ('Data Scientist/ ML Engineer', 'junior', 'de', 63500, 83000),
    ('Data Scientist/ ML Engineer', 'medior', 'de', 74500, 98500),
    ('Data Scientist/ ML Engineer', 'senior', 'de', 72000, 104000),
]

# Create the DataFrame
glassdoor_data = pd.DataFrame.from_records(glassdoor_data_rows, columns=columns)
```

In [19]:
```
# Define the column names
columns = ['job_category', 'seniority_level', 'country', 'glassdoor_lower', 'glassdoor_upper']

# Create a list of tuples
glassdoor_data_rows = [
    ('Data Analyst', 'junior', 'us', 70000, 117000),
    ('Data Analyst', 'medior', 'us', 81000, 134000),
    ('Data Analyst', 'senior', 'us', 88000, 150000),

    ('Data Analyst', 'junior', 'de', 51500, 70000),
    ('Data Analyst', 'medior', 'de', 55000, 71000),
    ('Data Analyst', 'senior', 'de', 68000, 93000),

    ('Data Engineer', 'junior', 'us', 84000, 142000),
    ('Data Engineer', 'medior', 'us', 96000, 157000),
    ('Data Engineer', 'senior', 'us', 121000, 198000),

    ('Data Engineer', 'junior', 'de', 57000, 74500),
    ('Data Engineer', 'medior', 'de', 62500, 81000),
    ('Data Engineer', 'senior', 'de', 73500, 94000),

    ('Data Scientist/ ML Engineer', 'junior', 'us', 117000, 196000),
    ('Data Scientist/ ML Engineer', 'medior', 'us', 129000, 210000),
    ('Data Scientist/ ML Engineer', 'senior', 'us', 149000, 251000),

    ('Data Scientist/ ML Engineer', 'junior', 'de', 58000, 79000),
    ('Data Scientist/ ML Engineer', 'medior', 'de', 63500, 83000),
    ('Data Scientist/ ML Engineer', 'senior', 'de', 72000, 104000),
]

# Create the DataFrame
glassdoor_data = pd.DataFrame.from_records(glassdoor_data_rows, columns=columns)
```

Exclusion categories

In [21]:
```
seniority_exclusion = ['other'] #executive

exclude_categories = ['Consultant', '"Other"', 'Uncategorized', 'Advocacy', 'Out of scope', 'Too vague answers', 'Other managers']
jobcategory_exclusion = ['Consultant', '"Other"', 'Uncategorized', 'Advocacy', 'Out of scope', 'Too vague answers']
```

In [22]:
```
# First we filter for western countries
df_ai_w= df_ai.copy()
```

```
df_it_w= df_it.copy()
df_k_w = df_k.copy()

# Western countries
df_ai_w = df_ai_w[df_ai_w['country'].isin(western_countries)].copy()
df_it_w = df_it_w[df_it_w['country'].isin(western_countries)].copy()
df_k_w = df_k_w[df_k_w['country'].isin(western_countries)].copy()

# Filter out rows where 'job_category' is in the exclude list
df_ai_w = df_ai_w[~df_ai_w['seniority_level'].isin(seniority_exclusion)]
df_it_w = df_it_w[~df_it_w['seniority_level'].isin(seniority_exclusion)]
df_k_w = df_k_w[~df_k_w['seniority_level'].isin(seniority_exclusion)]

df_ai_w = df_ai_w[~df_ai_w['job_category'].isin(jobcategory_exclusion)]
df_it_w = df_it_w[~df_it_w['job_category'].isin(jobcategory_exclusion)]
df_k_w = df_k_w[~df_k_w['job_category'].isin(jobcategory_exclusion)]
```

In [23]:
```
# Filtering out rows, which would result in Factorial Groups with less datapoint than the statistically required threshold

# Specify the minimum number of counts required to keep the group
min_count = 20

# Calculate the count of data points for each job category
group_counts1 = df_ai_w.groupby(['seniority_level', 'job_category'], observed=True)['salary_norm'].count().reset_index()
group_counts2 = df_it_w.groupby(['seniority_level', 'job_category'], observed=True)['salary_norm'].count().reset_index()
group_counts3 = df_k_w.groupby(['seniority_level', 'job_category'], observed=True)['salary_norm'].count().reset_index()

# Rename the count column for clarity
group_counts1.rename(columns={'salary_norm': 'count'}, inplace=True)
group_counts2.rename(columns={'salary_norm': 'count'}, inplace=True)
group_counts3.rename(columns={'salary_norm': 'count'}, inplace=True)

# Filter groups that meet the criteria
valid_groups1 = group_counts1[group_counts1['count'] >= min_count]
valid_groups2 = group_counts2[group_counts2['count'] >= min_count]
valid_groups3 = group_counts3[group_counts3['count'] >= min_count]

# Merge the valid groups back with the original dataframe to keep only the desired rows
df_ai_w_l = pd.merge(df_ai_w, valid_groups1[['seniority_level', 'job_category']],
                     on=['seniority_level', 'job_category'],
                     how='inner')

df_it_w_l = pd.merge(df_it_w, valid_groups2[['seniority_level', 'job_category']],
                     on=['seniority_level', 'job_category'],
                     how='inner')

df_k_w_l = pd.merge(df_k_w, valid_groups3[['seniority_level', 'job_category']],
                     on=['seniority_level', 'job_category'],
                     how='inner')
```

In [24]:
```
df_combined = pd.concat([df.dropna(axis=1, how='all') for df in [df_ai_w_l, df_it_w_l, df_k_w_l]])
```

Data-professionals

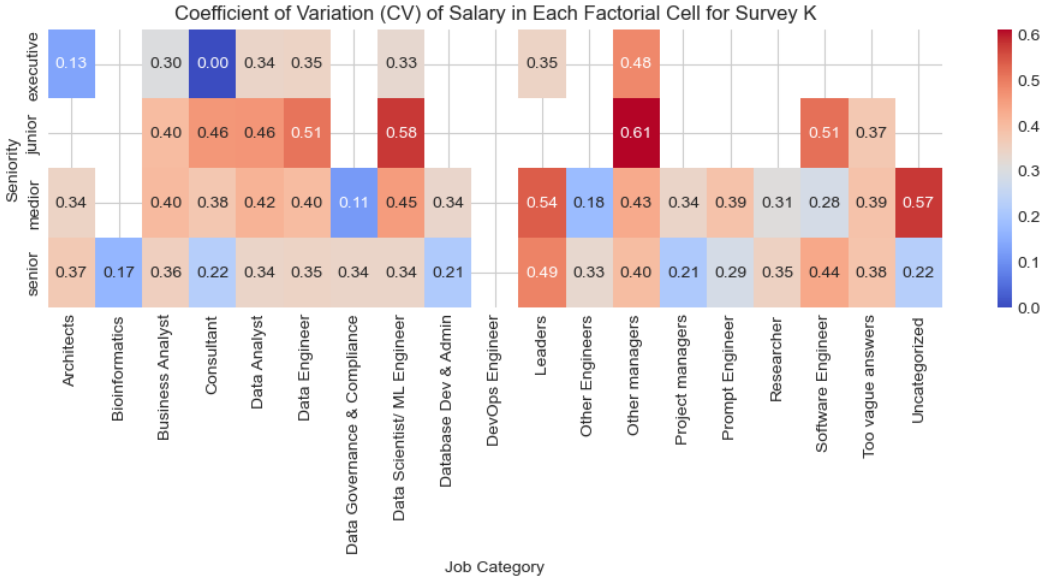
In [26]:
```
data_fields = ['Data Analyst', 'Data Engineer', 'Data Scientist/ ML Engineer']

df_k_w_data =  df_k_w[df_k_w['job_category'].isin(data_fields)]
df_it_w_data = df_it[df_it['job_category'].isin(data_fields)]
df_ai_w_data = df_ai_w[df_ai_w['job_category'].isin(data_fields)]
```

In [27]:
```
df_k_w_data = df_k_w_data[(df_k_w_data['seniority_level'] != 'executive')]
df_it_w_data = df_it_w_data[(df_it_w_data['seniority_level'] != 'executive')]
df_it_w_data = df_it_w_data[(df_it_w_data['seniority_level'] != 'other')]
df_ai_w_data = df_ai_w_data[(df_ai_w_data['seniority_level'] != 'executive')]
```

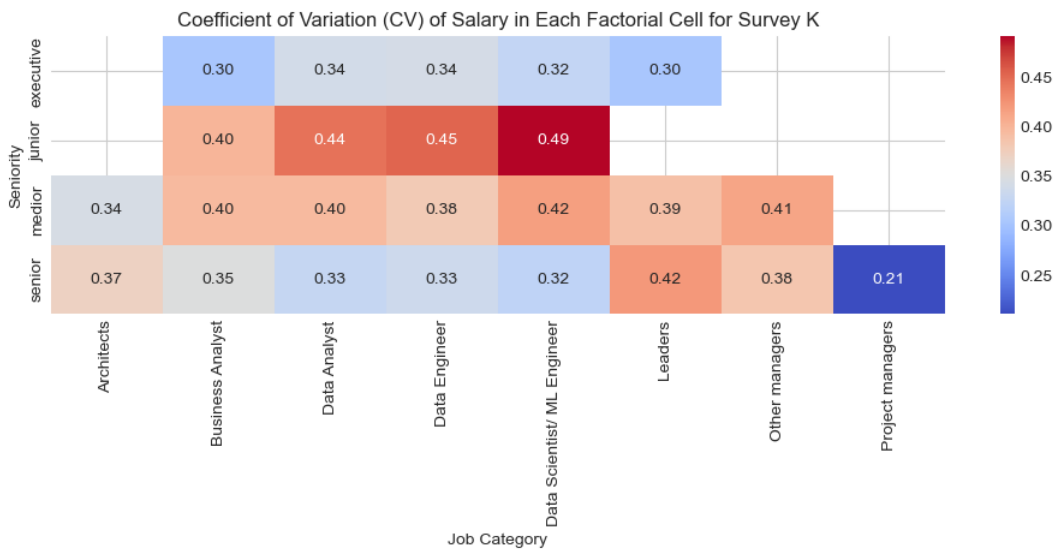
CV in each factorial cell to get a feeling for RMSE and R-squared

CV (Coefficient of Variation) = Standard Deviation / Mean salary

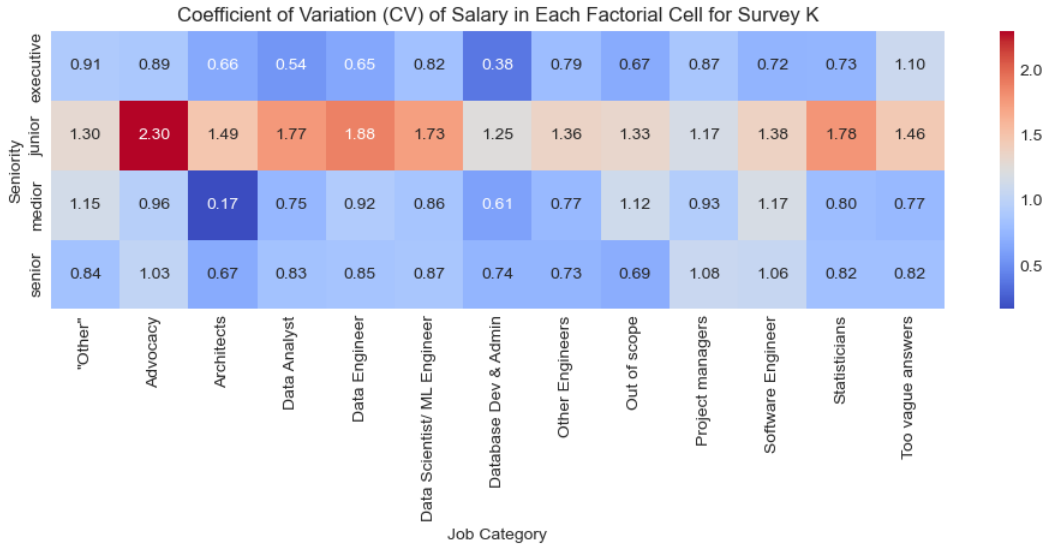
In [30]:
```
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_ai.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
pivot_table_std = df_ai.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(12, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```



In [31]:
```
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_ai_w_l.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
pivot_table_std = df_ai_w_l.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(12, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```

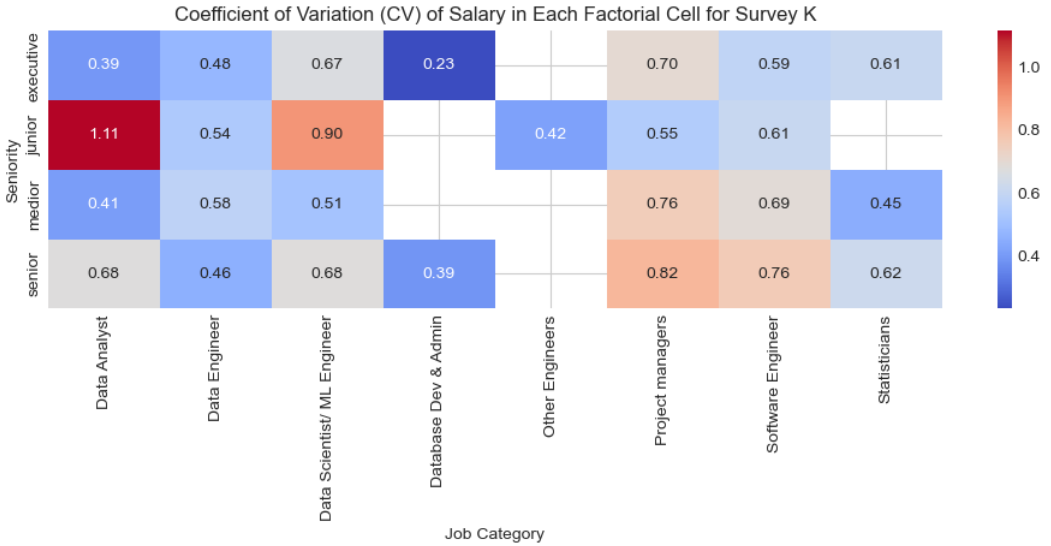


In [32]:
```python
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_k.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
pivot_table_std = df_k.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(12, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```



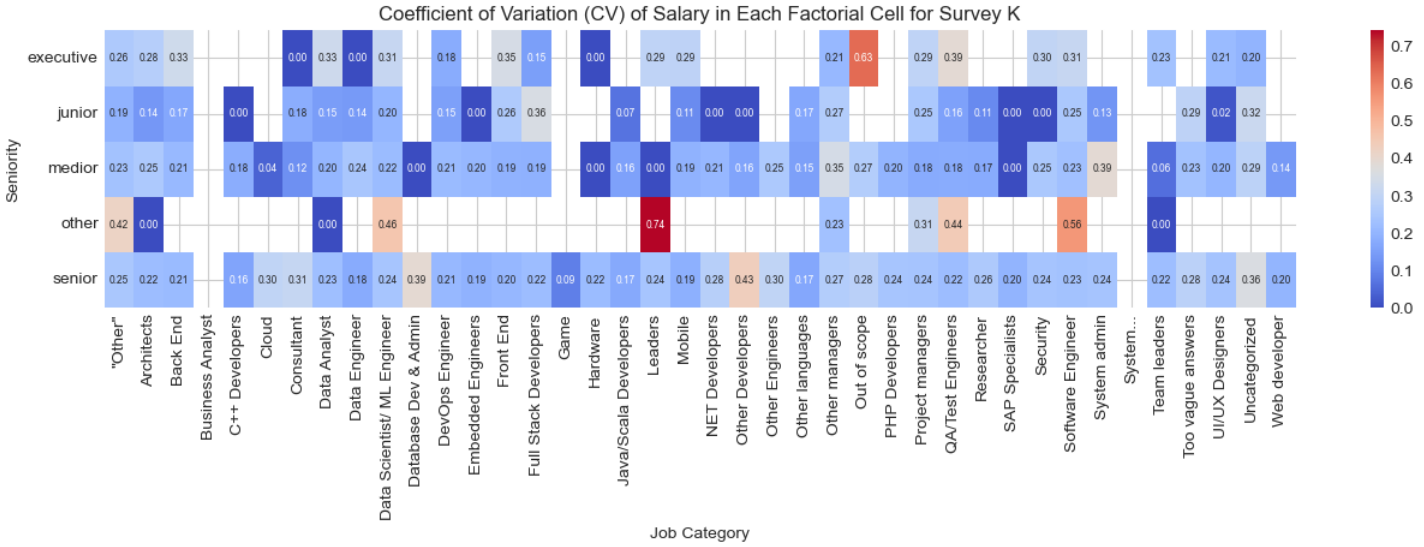
In [33]:
```python
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_k_w_1.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
```

```
pivot_table_std = df_k_w_l.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(12, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```



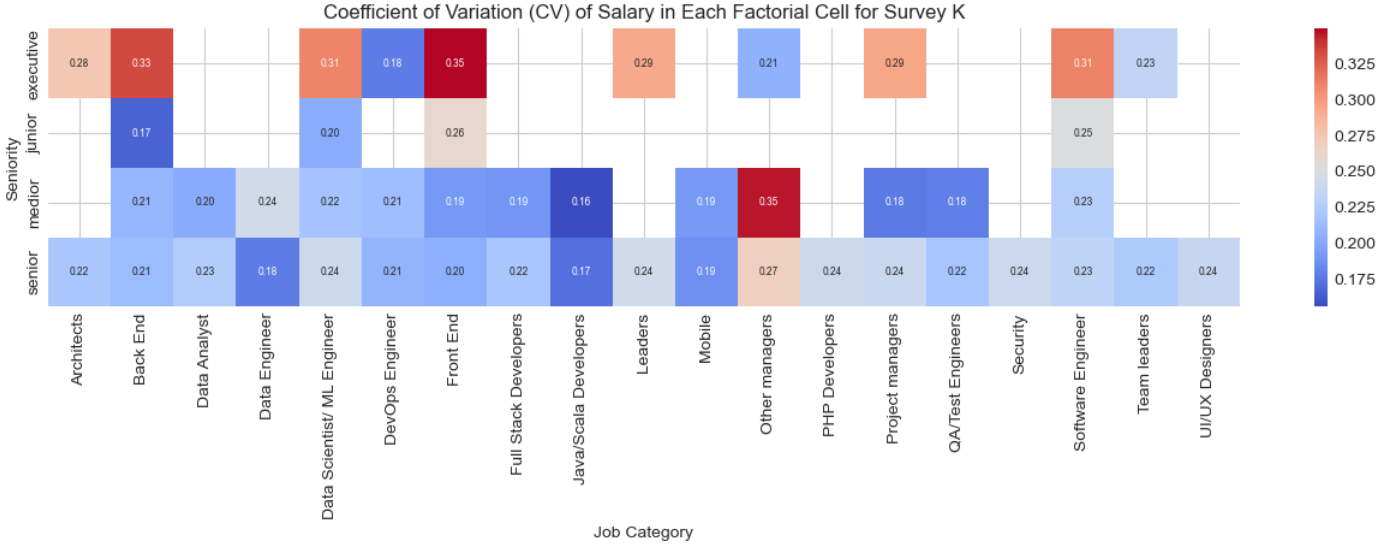
In [34]:

```
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_it.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
pivot_table_std = df_it.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(16, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm', annot_kws={"size": 6})
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```



In [35]:

```
# Create a pivot table to calculate the mean salary in each factorial cell
pivot_table_mean = df_it_w_l.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='mean', fill_value=0)

# Create a pivot table to calculate the standard deviation of salary in each factorial cell
pivot_table_std = df_it_w_l.pivot_table(index=['seniority_level'], columns='job_category', values='salary', aggfunc='std', fill_value=0)

# Calculate the Coefficient of Variation (CV) by dividing std by the mean
pivot_table_cv = pivot_table_std / pivot_table_mean

# Plot the heatmap of CV
plt.figure(figsize=(16, 3))
sns.heatmap(pivot_table_cv, annot=True, fmt='.2f', cmap='coolwarm', annot_kws={"size": 6})
plt.title('Coefficient of Variation (CV) of Salary in Each Factorial Cell for Survey K')
plt.xlabel('Job Category')
plt.ylabel('Seniority')
plt.show()
```



Multiregression

In [37]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


from patsy import dmatrices
from scipy.stats import shapiro
from scipy.stats import lognorm

import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Functions

In [39]:
```
def train_model_2301(X, y):
    """
    Trains a multilinear regression model using statsmodels, with one-hot encoding for categorical variables.
    Handles both numerical and categorical variables properly.

    Parameters:
    X (pd.DataFrame): Independent variables (can contain both categorical and numerical columns)
    y (pd.Series or pd.DataFrame): Dependent variable

    Returns:
    model: Fitted statsmodels regression model
    encoded_features: Dictionary containing feature names and their encodings for future use
    X_encoded: The encoded independent variables used in the model
    y: The dependent variable aligned with X_encoded
    """
    # Create copies to avoid modifying the original data
    X_copy = X.copy()
    y_copy = y.copy()

    # Dictionary to store category mappings
    encoded_features = {}

    # Process each column
    for column in X_copy.columns:
        # Check if column is categorical (object dtype) or contains strings
        if X_copy[column].dtype == 'object' or pd.api.types.is_string_dtype(X_copy[column]):
            # Store unique categories for this column
            encoded_features[column] = sorted(X_copy[column].unique().tolist())

    # One-hot encode categorical variables
    X_encoded = pd.get_dummies(X_copy, drop_first=True, dtype=float)

    # Add constant term
    X_encoded = sm.add_constant(X_encoded)

    # Convert y to float type if it isn't already
    y_copy = y_copy.astype(float)

    # Reset indices to ensure alignment
    X_encoded.reset_index(drop=True, inplace=True)
    y_copy.reset_index(drop=True, inplace=True)

    # Fit the model
    model = sm.OLS(y_copy, X_encoded).fit()

    return model, encoded_features, X_encoded, y_copy
```

In [40]:
```
import pandas as pd
import statsmodels.api as sm

def train_model_1617(X, y):
```

```
    """
    Trains a multilinear regression model using statsmodels, with one-hot encoding for categorical variables.
    Handles both numerical and categorical variables properly.

    Parameters:
    X (pd.DataFrame): Independent variables (can contain both categorical and numerical columns)
    y (pd.Series or pd.DataFrame): Dependent variable

    Returns:
    model: Fitted statsmodels regression model
    encoded_features: Dictionary containing feature names and their encodings for future use
    X_encoded: The encoded independent variables used in the model
    y: The dependent variable aligned with X_encoded
    """
    # Create copies to avoid modifying the original data
    X_copy = X.copy()
    y_copy = y.copy()

    # Dictionary to store category mappings
    encoded_features = {}

    # Process each column
    for column in X_copy.columns:
        # Check if column is categorical (object dtype) or contains strings
        if X_copy[column].dtype == 'object' or pd.api.types.is_string_dtype(X_copy[column]):
            # Fill NaN values with a placeholder
            X_copy[column] = X_copy[column].fillna('Missing')
            # Store unique categories for this column
            encoded_features[column] = sorted(X_copy[column].unique().tolist())

    # One-hot encode categorical variables
    X_encoded = pd.get_dummies(X_copy, drop_first=True, dtype=float)

    # Add constant term
    X_encoded = sm.add_constant(X_encoded)

    # Convert y to float type if it isn't already
    y_copy = y_copy.astype(float)

    # Reset indices to ensure alignment
    X_encoded.reset_index(drop=True, inplace=True)
    y_copy.reset_index(drop=True, inplace=True)

    # Fit the model
    model = sm.OLS(y_copy, X_encoded).fit()

    return model, encoded_features, X_encoded, y_copy
```

In [41]:

```
# Function to calculate Leverage and Cook's Distance
def calculate_leverage_and_cooks_distance(model):
    # Get influence measures
    influence = model.get_influence()
    leverage = influence.hat_matrix_diag
    cooks_d = influence.cooks_distance[0]
    return leverage, cooks_d

# Function to plot Leverage and Cook's Distance
def plot_leverage_and_cooks_distance(leverage, cooks_d, leverage_threshold=None, cooks_threshold=None):
    n = len(leverage)
    p = model.df_model  # Number of predictors (including dummy variables)

    if leverage_threshold is None:
        leverage_threshold = 2 * (p + 1) / n
    if cooks_threshold is None:
        cooks_threshold = 4 / n

    fig, ax = plt.subplots(1, 2, figsize=(14, 3))

    # Leverage Plot
    ax[0].scatter(range(n), leverage, alpha=0.5, s=5)
    ax[0].axhline(y=leverage_threshold, color='r', linestyle='--', label='Leverage Threshold')
    ax[0].set_ylim(0,0.06)
    ax[0].set_xlabel('Observation')
    ax[0].set_ylabel('Leverage')
    ax[0].set_title('Leverage Values')
    ax[0].legend()
```

```
    # Cook's Distance Plot
    ax[1].scatter(range(n), cooks_d, alpha=0.5, s=5)
    ax[1].axhline(y=cooks_threshold, color='r', linestyle='--', label="Cook's Distance Threshold")
    ax[1].set_ylim(0,0.005)
    ax[1].set_xlabel('Observation')
    ax[1].set_ylabel("Cook's Distance")
    ax[1].set_title("Cook's Distance Values")
    ax[1].legend()

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

In [42]:

```
def plot_model_coefficients(model, sizex, sizey):
    """
    Extracts and plots the coefficients from a statsmodels OLS regression model, excluding the intercept.
    Parameters:
    model (statsmodels.regression.linear_model.RegressionResultsWrapper): The fitted OLS model.
    """
    # Extract the coefficients and their names
    coefficients = model.params
    feature_names = coefficients.index

    # Create a DataFrame for coefficients
    coefficients_df = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': coefficients.values
    })

    # Remove the constant (intercept) term if it exists
    coefficients_df = coefficients_df[coefficients_df['Feature'] != 'const']

    # Sort the dataframe by coefficient values for better visualization
    coefficients_df_sorted = coefficients_df.sort_values(by='Coefficient')

    # Create a horizontal bar plot
    plt.figure(figsize=(sizex, sizey))
    sns.barplot(
        x='Coefficient',
        y='Feature',
        data=coefficients_df_sorted,
        hue='Feature',
        palette='coolwarm',
        orient='h',
        legend=False
    )

    # Add a vertical line at x=0 for reference
    plt.axvline(x=0, color='grey', linewidth=0.8)

    # Set plot title and labels
    plt.title('Regression Coefficients for Salary Prediction Model')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Features')

    # Adjust layout to ensure all labels are visible
    plt.tight_layout()

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

In [43]:

```
def calculate_rmse_original_scale(model, y_true_log):
    """
    Calculates RMSE on the original scale for a model trained with log-transformed values.

    Parameters:
    - model: The fitted statsmodels OLS model.
    - y_true_log: The actual log-transformed values of the dependent variable.

    Returns:
    - rmse_original: RMSE calculated on the original (non-logarithmic) scale.
    """
    # Calculate predictions in the log scale
    y_pred_log = model.fittedvalues

    # Back-transform predictions and actuals to the original scale
    y_pred = np.exp(y_pred_log)
    y_true = np.exp(y_true_log)
```

```
    # Calculate RMSE in the original scale
    rmse_original = np.sqrt(np.mean((y_true - y_pred) ** 2))

    return rmse_original
```

In [44]:
```
def predict_salary_for_case_log(model, encoded_features, specific_case):
    """
    Predicts the salary for a specific case using the statsmodels OLS model.

    Parameters:
    - model: The fitted statsmodels OLS model.
    - encoded_features: Dictionary containing feature names and their encodings.
    - specific_case: Dictionary of feature values for the specific case.

    Returns:
    - predicted_salary: The predicted salary (transformed back from log scale if applicable).
    """
    # Create a DataFrame for the specific case
    specific_case_df = pd.DataFrame([specific_case])

    # Initialize all features with zero values
    all_features = model.params.index.tolist()
    specific_case_encoded = pd.DataFrame(columns=all_features)
    specific_case_encoded.loc[0] = 0  # Initialize all features to zero

    # Handle categorical variables
    for column in encoded_features.keys():
        categories = encoded_features[column]
        # Skip the reference category due to drop_first=True
        for cat in categories[1:]:
            col_name = f"{column}_{cat}"
            if specific_case[column] == cat:
                specific_case_encoded.at[0, col_name] = 1
            else:
                specific_case_encoded.at[0, col_name] = 0

    # Handle numerical variables (if any)
    numerical_columns = [col for col in specific_case.keys() if col not in encoded_features.keys()]
    for col in numerical_columns:
        specific_case_encoded.at[0, col] = specific_case[col]

    # Add constant term
    specific_case_encoded['const'] = 1.0  # Ensure the constant term is included

    # Reorder columns to match the model's parameters
    specific_case_encoded = specific_case_encoded[model.params.index]

    # Ensure all columns are float64
    specific_case_encoded = specific_case_encoded.astype('float64')

    # Make the prediction
    predicted_log_salary = model.predict(specific_case_encoded)[0]

    # If the model was trained on log-transformed salaries, exponentiate to get the actual salary
    # Adjust this line if your target variable is not log-transformed
    predicted_salary = np.exp(predicted_log_salary)

    return predicted_salary
```

In [45]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm

def predict_salary_for_case_log_perc(model, encoded_features, specific_case):
    """
    Predicts the salary and calculates the 25th and 75th percentiles for a specific case
    using the statsmodels OLS model trained on log-transformed salaries.

    Parameters:
    - model: The fitted statsmodels OLS model.
    - encoded_features: Dictionary containing feature names and their encodings.
    - specific_case: Dictionary of feature values for the specific case.

    Returns:
    - predicted_salary: The predicted median salary (original scale).
```

```
    - p25_salary: The 25th percentile of the predicted salary distribution (original scale).
    - p75_salary: The 75th percentile of the predicted salary distribution (original scale).
    """
    # Create a DataFrame for the specific case
    specific_case_df = pd.DataFrame([specific_case])

    # Initialize all features with zero values
    all_features = model.params.index.tolist()
    specific_case_encoded = pd.DataFrame(columns=all_features)
    specific_case_encoded.loc[0] = 0  # Initialize all features to zero

    # Handle categorical variables
    for column in encoded_features.keys():
        categories = encoded_features[column]
        # Skip the reference category due to drop_first=True
        for cat in categories[1:]:
            col_name = f"{column}_{cat}"
            if specific_case[column] == cat:
                specific_case_encoded.at[0, col_name] = 1
            else:
                specific_case_encoded.at[0, col_name] = 0

    # Handle numerical variables (if any)
    numerical_columns = [col for col in specific_case.keys() if col not in encoded_features.keys()]
    for col in numerical_columns:
        specific_case_encoded.at[0, col] = specific_case[col]

    # Add constant term
    specific_case_encoded['const'] = 1.0  # Ensure the constant term is included

    # Reorder columns to match the model's parameters
    specific_case_encoded = specific_case_encoded[model.params.index]

    # Ensure all columns are float64
    specific_case_encoded = specific_case_encoded.astype('float64')

    # Make the prediction in log scale
    predicted_log_salary = model.predict(specific_case_encoded)[0]

    # Obtain the standard deviation of the residuals (sigma_log)
    sigma_log = np.sqrt(model.scale)

    # Compute the 25th and 75th percentiles in log space
    p25_log_salary = predicted_log_salary + norm.ppf(0.25) * sigma_log
    p75_log_salary = predicted_log_salary + norm.ppf(0.75) * sigma_log

    # Exponentiate to get the salaries in original scale
    predicted_salary = np.exp(predicted_log_salary)
    p25_salary = np.exp(p25_log_salary)
    p75_salary = np.exp(p75_log_salary)

    return predicted_salary, p25_salary, p75_salary
```

In [46]:

```
def predict_salary_for_case(model, encoded_features, specific_case):
    """
    Predicts the salary for a specific case using the statsmodels OLS model.

    Parameters:
    - model: The fitted statsmodels OLS model.
    - encoded_features: Dictionary containing feature names and their encodings.
    - specific_case: Dictionary of feature values for the specific case.

    Returns:
    - predicted_salary: The predicted salary (transformed back from log scale if applicable).
    """
    # Create a DataFrame for the specific case
    specific_case_df = pd.DataFrame([specific_case])

    # Initialize all features with zero values
    all_features = model.params.index.tolist()
    specific_case_encoded = pd.DataFrame(columns=all_features)
    specific_case_encoded.loc[0] = 0  # Initialize all features to zero

    # Handle categorical variables
    for column in encoded_features.keys():
        categories = encoded_features[column]
        # Skip the reference category due to drop_first=True
```

```
        for cat in categories[1:]:
            col_name = f"{column}_{cat}"
            if specific_case[column] == cat:
                specific_case_encoded.at[0, col_name] = 1
            else:
                specific_case_encoded.at[0, col_name] = 0

    # Handle numerical variables (if any)
    numerical_columns = [col for col in specific_case.keys() if col not in encoded_features.keys()]
    for col in numerical_columns:
        specific_case_encoded.at[0, col] = specific_case[col]

    # Add constant term
    specific_case_encoded['const'] = 1.0  # Ensure the constant term is included

    # Reorder columns to match the model's parameters
    specific_case_encoded = specific_case_encoded[model.params.index]

    # Ensure all columns are float64
    specific_case_encoded = specific_case_encoded.astype('float64')

    # Make the prediction
    predicted_salary = model.predict(specific_case_encoded)[0]

    return predicted_salary
```

Plotting functions

In [48]:
```
def plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6):
    """
    Plots a histogram with KDE and a Q-Q plot of residuals side by side.

    Parameters:
        residuals (pd.Series): Series of residuals to analyze.
        hist_bins (int): Number of bins for the histogram.
        qq_marker_size (float): Size of the dots in the Q-Q plot.
        qq_alpha (float): Alpha (transparency) of the dots in the Q-Q plot.
    """
    # Create a figure with two subplots
    fig, axes = plt.subplots(1, 2, figsize=(14, 3))

    # Plot histogram with KDE
    sns.histplot(residuals, kde=True, bins=hist_bins, ax=axes[0])
    axes[0].set_title('Histogram of Residuals')
    axes[0].set_xlabel('Residual')
    axes[0].set_ylabel('Frequency')

    # Create Q-Q plot
    qq_plot = sm.qqplot(residuals, line='s', ax=axes[1])
    plt.setp(qq_plot.gca().get_lines(), markersize=qq_marker_size, alpha=qq_alpha)
    axes[1].set_title('Q-Q Plot of Residuals')
    axes[1].set_xlabel('Theoretical Quantiles')
    axes[1].set_ylabel('Sample Quantiles')

    # Adjust layout and display
    plt.tight_layout()
    plt.show()
```

In [49]:
```
def simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=1000, dot_size=10):
    """
    Simulates Shapiro-Wilk tests on random subsamples of residuals over a range of sample sizes,
    calculating the average p-value across simulations for each sample size.

    Parameters:
        residuals (pd.Series): Series of residuals to sample from.
        min_sample_size (int): Minimum sample size for the subsampling.
        max_sample_size (int): Maximum sample size for the subsampling.
        step_size (int): Step size for increasing sample size.
        num_simulations (int): Number of simulations per sample size.
        dot_size (int): Size of the dots in the final plot.

    Returns:
        pd.DataFrame: DataFrame containing sample sizes and corresponding average Shapiro-Wilk p-values.
    """
```

```
    # Define the range of sample sizes
    sample_sizes = np.arange(min_sample_size, max_sample_size + 1, step_size)

    # Initialize a list to store the results
    results = []

    # Loop through each sample size
    for size in sample_sizes:
        p_values = []  # List to store p-values for each simulation at this sample size

        # Perform multiple simulations for each sample size
        for _ in range(num_simulations):
            # Take a random subsample of the given size from residuals
            subsample = residuals.sample(size, random_state=np.random.randint(0, 10000))

            # Perform Shapiro-Wilk test and store the p-value
            _, p_value = shapiro(subsample)
            p_values.append(p_value)

        # Calculate the average p-value across simulations
        avg_p_value = np.mean(p_values)

        # Append results to the list
        results.append({'Sample Size': size, 'Average Shapiro-Wilk p-value': avg_p_value})

    # Convert results to a DataFrame for easy plotting
    results_df = pd.DataFrame(results)

    # Plot the Sample Size vs. Average P-Value curve
    plt.figure(figsize=(12, 4))
    plt.scatter(results_df['Sample Size'], results_df['Average Shapiro-Wilk p-value'], alpha=0.5, s=dot_size, c="blue")
    plt.axhline(0.05, color='red', linestyle='--', label='p=0.05 threshold')
    plt.axhline(0.10, color='orange', linestyle='--', label='p=0.10 threshold')
    plt.xlabel('Sample Size')
    plt.ylabel('Average Shapiro-Wilk p-value')
    plt.title('Sample Size vs. Average Shapiro-Wilk p-value (across simulations)')
    plt.legend()
    plt.show()

    return results_df
```

In [50]:
```
# Plot residuals to check for homoscedasticity
def plot_homoscedasticity(residuals):

    # Scatter plot of residuals vs fitted values
    plt.scatter(fitted_values, residuals, alpha=0.05, s=5, color='purple')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Fitted Values')
    plt.ylabel('Residuals')
    plt.title('Residuals vs Fitted Values')
    plt.show()
```

In [51]:
```
# Function to plot a scale-location plot
def plot_scale_location(fitted_values, residuals):
    """
    Plots a scale-location plot to check for homoscedasticity.

    Parameters:
    fitted_values (array-like): Fitted values from the regression model
    residuals (array-like): Residuals from the regression model
    """
    # Calculate standardized residuals
    standardized_residuals = residuals / residuals.std()

    # Compute square root of standardized residuals
    sqrt_standardized_residuals = np.sqrt(np.abs(standardized_residuals))

    # Scatter plot of square root of standardized residuals vs fitted values
    plt.scatter(fitted_values, sqrt_standardized_residuals, alpha=0.5, s=10, color='blue')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Fitted Values')
    plt.ylabel('√|Standardized Residuals|')
    plt.title('Scale-Location Plot')
    plt.show()
```

In [52]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
def plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=10, alpha=0.5):
    """
    Creates two plots side by side:
    1. Residuals vs Fitted Values plot
    2. Scale-Location plot (sqrt of standardized residuals vs fitted values).

    Parameters:
    fitted_values (array-like): Fitted values from the regression model
    residuals (array-like): Residuals from the regression model
    dot_size (int, optional): Size of the scatter plot dots. Default is 10.
    alpha (float, optional): Transparency of the scatter plot dots. Default is 0.5.
    """
    # Calculate standardized residuals
    standardized_residuals = residuals / residuals.std()

    # Compute square root of standardized residuals
    sqrt_standardized_residuals = np.sqrt(np.abs(standardized_residuals))

    # Set up the side-by-side plot layout
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))

    # Plot 1: Residuals vs Fitted Values
    axes[0].scatter(fitted_values, residuals, alpha=alpha, s=dot_size, color='purple')
    axes[0].axhline(y=0, color='r', linestyle='--')
    axes[0].set_xlabel('Fitted Values')
    axes[0].set_ylabel('Residuals')
    axes[0].set_title('Residuals vs Fitted Values')

    # Plot 2: Scale-Location Plot
    axes[1].scatter(fitted_values, sqrt_standardized_residuals, alpha=alpha, s=dot_size, color='purple')
    axes[1].axhline(y=0, color='r', linestyle='--')
    axes[1].set_xlabel('Fitted Values')
    axes[1].set_ylabel('√|Standardized Residuals|')
    axes[1].set_title('Scale-Location Plot')

    # Adjust spacing
    plt.tight_layout()
    plt.show()
```

In [53]:

```
def simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=17000, step_size=100, num_simulations=100, dot_size=10):
    """
    Simulates Breusch-Pagan tests on random subsamples of residuals and fitted values over a range of sample sizes,
    calculating the average p-value across simulations for each sample size.

    Parameters:
        residuals (pd.Series): Series of residuals to sample from.
        fitted_values (pd.Series): Series of fitted values corresponding to the residuals.
        min_sample_size (int): Minimum sample size for the subsampling.
        max_sample_size (int): Maximum sample size for the subsampling.
        step_size (int): Step size for increasing sample size.
        num_simulations (int): Number of simulations per sample size.
        dot_size (int): Size of the dots in the final plot.

    Returns:
        pd.DataFrame: DataFrame containing sample sizes and corresponding average Breusch-Pagan p-values.
    """
    # Define the range of sample sizes
    sample_sizes = np.arange(min_sample_size, max_sample_size + 1, step_size)

    # Initialize a list to store the results
    results = []

    # Loop through each sample size
    for size in sample_sizes:
        p_values = []  # List to store p-values for each simulation at this sample size

        # Perform multiple simulations for each sample size
        for _ in range(num_simulations):
            # Randomly sample indices and select subset of residuals and fitted values
            indices = np.random.choice(range(len(residuals)), size=size, replace=False)
            subsample_residuals = residuals.iloc[indices]
            subsample_fitted_values = fitted_values.iloc[indices]

            # Prepare data for Breusch-Pagan test
            exog = add_constant(subsample_fitted_values)  # Add constant to independent variables
            bp_test = het_breuschpagan(subsample_residuals, exog)
```

```
            # Store the p-value from the Breusch-Pagan test
            p_value = bp_test[1]  # p-value is the second element in the returned tuple
            p_values.append(p_value)

        # Calculate the average p-value across simulations
        avg_p_value = np.mean(p_values)

        # Append results to the list
        results.append({'Sample Size': size, 'Average Breusch-Pagan p-value': avg_p_value})

    # Convert results to a DataFrame for easy plotting
    results_df = pd.DataFrame(results)

    # Plot the Sample Size vs. Average P-Value curve
    plt.figure(figsize=(10, 4))
    plt.scatter(results_df['Sample Size'], results_df['Average Breusch-Pagan p-value'], alpha=0.5, s=dot_size, c="purple")
    plt.axhline(0.05, color='red', linestyle='--', label='p=0.05 threshold')
    plt.axhline(0.10, color='orange', linestyle='--', label='p=0.10 threshold')
    plt.xlabel('Sample Size')
    plt.ylabel('Average Breusch-Pagan p-value')
    plt.title('Sample Size vs. Average Breusch-Pagan p-value (across simulations)')
    plt.legend()
    plt.show()

    return results_df
```

In [54]:

```
def plot_salary_distributions(df_name, model, encoded_features, specific_case, glassdoor_data):
    """
    Plots predicted and fitted lognormal salary distributions, observed salary data,
    and Glassdoor salary range (if available) for a specific case.

    Parameters:
        df_name (pd.DataFrame): DataFrame containing observed salary data.
        model: Model for predicting salary.
        encoded_features: Encoded features for prediction.
        specific_case (dict): Dictionary with keys 'job_category', 'seniority_level', and 'country'.
        glassdoor_data (pd.DataFrame): DataFrame with Glassdoor salary range data.

    Returns:
        None
    """
    # Extract case parameters
    job_category = specific_case['job_category']
    seniority_level = specific_case['seniority_level']
    country = specific_case['country']

    # Retrieve salary data for specific case
    salary_data = df_name['salary'][
        (df_name['seniority_level'] == seniority_level) &
        (df_name['job_category'] == job_category) &
        (df_name['country'] == country)
    ]

    # Fit a lognormal distribution to observed salary data
    shape_fit, loc_fit, scale_fit = lognorm.fit(salary_data, floc=0)
    mu_fit = np.log(scale_fit)
    sigma_fit = shape_fit

    # Predict salary and percentiles
    predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(
        model, encoded_features, specific_case
    )

    # Calculate predicted lognormal parameters
    z_25 = -0.67448975  # z-score for the 25th percentile
    z_75 = 0.67448975   # z-score for the 75th percentile
    sigma_pred = (np.log(predicted_salary_p75) - np.log(predicted_salary_p25)) / (z_75 - z_25)
    mu_pred = np.log(predicted_salary)

    # Set upper limit for x-axis
    P99_pred = lognorm.ppf(0.99, s=sigma_pred, scale=np.exp(mu_pred))
    P99_fitted = lognorm.ppf(0.99, s=sigma_fit, scale=np.exp(mu_fit))
    upper_limit = max(P99_pred, P99_fitted, salary_data.max()) * 1.1
    x = np.linspace(0.01, upper_limit, 200)

    # Generate lognormal PDFs
```

```
    predicted_pdf = lognorm.pdf(x, s=sigma_pred, scale=np.exp(mu_pred))
    fitted_pdf = lognorm.pdf(x, s=sigma_fit, scale=np.exp(mu_fit))

    # Calculate percentiles for predicted and fitted distributions
    P25_pred, P50_pred, P75_pred = predicted_salary_p25, predicted_salary, predicted_salary_p75
    P25_fitted = lognorm.ppf(0.25, s=sigma_fit, scale=np.exp(mu_fit))
    P50_fitted = lognorm.ppf(0.50, s=sigma_fit, scale=np.exp(mu_fit))
    P75_fitted = lognorm.ppf(0.75, s=sigma_fit, scale=np.exp(mu_fit))

    # Retrieve Glassdoor Salary Range
    glassdoor_row = glassdoor_data[
        (glassdoor_data['job_category'] == job_category) &
        (glassdoor_data['seniority_level'] == seniority_level) &
        (glassdoor_data['country'] == country)
    ]
    glassdoor_lower = glassdoor_row['glassdoor_lower'].values[0] if not glassdoor_row.empty else None
    glassdoor_upper = glassdoor_row['glassdoor_upper'].values[0] if not glassdoor_row.empty else None

    # Plotting
    plt.figure(figsize=(14, 6))
    plt.plot(x, predicted_pdf, label=f'Predicted Lognormal Distribution\n$\\mu={mu_pred:.2f}, \\sigma={sigma_pred:.2f}$', color='green')
    plt.plot(x, fitted_pdf, label=f'Fitted Lognormal Distribution\n$\\mu={mu_fit:.2f}, \\sigma={sigma_fit:.2f}$', color='blue')
    plt.hist(salary_data, bins=30, density=True, alpha=0.1, label='Observed Salary Data', color='blue')

    # Annotate percentiles for predicted distribution
    plt.axvline(P25_pred, color='green', alpha=0.7, linestyle=':', label=f'Predicted 25th percentile: {P25_pred:.2f}')
    plt.axvline(P50_pred, color='green', alpha=0.7, linestyle=':', label=f'Predicted 50th percentile (median): {P50_pred:.2f}')
    plt.axvline(P75_pred, color='green', alpha=0.7, linestyle=':', label=f'Predicted 75th percentile: {P75_pred:.2f}')

    # Annotate percentiles for fitted distribution
    plt.axvline(P25_fitted, color='blue', alpha=0.5, linestyle='--', label=f'Fitted 25th percentile: {P25_fitted:.2f}')
    plt.axvline(P50_fitted, color='blue', alpha=0.5, linestyle='--', label=f'Fitted 50th percentile (median): {P50_fitted:.2f}')
    plt.axvline(P75_fitted, color='blue', alpha=0.5, linestyle='--', label=f'Fitted 75th percentile: {P75_fitted:.2f}')

    # Add Glassdoor salary range if available
    if glassdoor_lower is not None and glassdoor_upper is not None:
        plt.axvspan(glassdoor_lower, glassdoor_upper, color='orange', alpha=0.25,
                    label=f'Glassdoor Salary Range: {glassdoor_lower} - {glassdoor_upper}')

    # Labels and title
    plt.title('Lognormal Distributions (Predicted and Fitted) and Observed Salary Data')
    plt.xlabel('Salary')
    plt.ylabel('Probability Density Function (PDF)')
    plt.legend()
    plt.grid(True)
    plt.show()
```

In [55]:

```
def plot_salary_percentiles_by_case(df_name, model, encoded_features, specific_cases, glassdoor_data):
    """
    Plots salary percentiles (P25, P50, P75) for different cases and data sources (Observed, Predicted, Glassdoor).

    Parameters:
        df_name (pd.DataFrame): DataFrame containing observed salary data.
        model: Model for predicting salary.
        encoded_features: Encoded features for prediction.
        specific_cases (list of dict): List of specific cases to analyze, with keys 'job_category', 'seniority_level', and 'country'.
        glassdoor_data (pd.DataFrame): DataFrame with Glassdoor salary range data.

    Returns:
        None
    """
    # Initialize lists to store data
    case_labels = []
    observed_percentiles = []
    predicted_percentiles = []
    glassdoor_percentiles = []

    for specific_case in specific_cases:
        job_category = specific_case['job_category']
        seniority_level = specific_case['seniority_level']
        country = specific_case['country']

        # --- Observed Data ---
        salary_data = df_name['salary'][
            (df_name['seniority_level'] == seniority_level) &
            (df_name['job_category'] == job_category) &
            (df_name['country'] == country)
```

```
    ]

    if len(salary_data) > 0:
        P25_obs = np.percentile(salary_data, 25)
        P50_obs = np.percentile(salary_data, 50)
        P75_obs = np.percentile(salary_data, 75)
    else:
        P25_obs = P50_obs = P75_obs = np.nan

    observed_percentiles.append([P25_obs, P50_obs, P75_obs])

    # --- Predicted Data ---
    predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(
        model, encoded_features, specific_case
    )

    P25_pred, P50_pred, P75_pred = predicted_salary_p25, predicted_salary, predicted_salary_p75
    predicted_percentiles.append([P25_pred, P50_pred, P75_pred])

    # --- Glassdoor Data ---
    glassdoor_row = glassdoor_data[
        (glassdoor_data['job_category'] == job_category) &
        (glassdoor_data['seniority_level'] == seniority_level) &
        (glassdoor_data['country'] == country)
    ]

    if not glassdoor_row.empty:
        glassdoor_lower = glassdoor_row['glassdoor_lower'].values[0]
        glassdoor_upper = glassdoor_row['glassdoor_upper'].values[0]
        glassdoor_median = (glassdoor_lower + glassdoor_upper) / 2
        P25_glassdoor, P50_glassdoor, P75_glassdoor = glassdoor_lower, glassdoor_median, glassdoor_upper
    else:
        P25_glassdoor = P50_glassdoor = P75_glassdoor = np.nan

    glassdoor_percentiles.append([P25_glassdoor, P50_glassdoor, P75_glassdoor])

    # Add case label
    case_label = f"{seniority_level.capitalize()} + {job_category} + {country.upper()}"
    case_labels.append(case_label)

# Create DataFrame for plotting
data_list = []
for idx, case_label in enumerate(case_labels):
    data_list.append({'Case': case_label, 'Source': 'Observed', 'P25': observed_percentiles[idx][0], 'P50': observed_percentiles[idx][1], 'P75': observed_percentiles[idx][2]})
    data_list.append({'Case': case_label, 'Source': 'Predicted', 'P25': predicted_percentiles[idx][0], 'P50': predicted_percentiles[idx][1], 'P75': predicted_percentiles[idx][2]})
    data_list.append({'Case': case_label, 'Source': 'Glassdoor', 'P25': glassdoor_percentiles[idx][0], 'P50': glassdoor_percentiles[idx][1], 'P75': glassdoor_percentiles[idx][2]})

df_plot = pd.DataFrame(data_list)
df_plot.sort_values(['Case', 'Source'], inplace=True)

# Plotting
cases = df_plot['Case'].unique()
sources = ['Observed', 'Predicted', 'Glassdoor']
n_cases = len(cases)
n_sources = len(sources)
bar_width = 0.2
x = np.arange(n_cases)
offsets = np.linspace(-bar_width * (n_sources - 1) / 2, bar_width * (n_sources - 1) / 2, n_sources)
source_colors = {'Observed': 'blue', 'Predicted': 'green', 'Glassdoor': 'orange'}

fig, ax = plt.subplots(figsize=(14, 6))
plotted_labels = set()

for i, source in enumerate(sources):
    bar_positions = x + offsets[i]
    medians, lower_errors, upper_errors = [], [], []

    for case in cases:
        source_data = df_plot[(df_plot['Case'] == case) & (df_plot['Source'] == source)]
        if not source_data.empty:
            median = source_data['P50'].values[0]
            P25 = source_data['P25'].values[0]
            P75 = source_data['P75'].values[0]
            lower_error = median - P25
            upper_error = P75 - median
        else:
            median = np.nan
```

```
                lower_error = upper_error = np.nan
            medians.append(median)
            lower_errors.append(lower_error)
            upper_errors.append(upper_error)

        # Convert to numpy arrays and plot
        medians, lower_errors, upper_errors = np.array(medians), np.array(lower_errors), np.array(upper_errors)
        valid = ~np.isnan(medians)

        if valid.any():
            if source not in plotted_labels:
                ax.bar(bar_positions[valid], medians[valid], width=bar_width, label=source,
                       color=source_colors.get(source), yerr=[lower_errors[valid], upper_errors[valid]], capsize=5)
                plotted_labels.add(source)
            else:
                ax.bar(bar_positions[valid], medians[valid], width=bar_width,
                       color=source_colors.get(source), yerr=[lower_errors[valid], upper_errors[valid]], capsize=5)

    ax.set_xticks(x)
    ax.set_xticklabels(cases, rotation=45, ha='right')
    ax.set_xlabel('Cases')
    ax.set_ylabel('Salary')
    ax.set_title('Salary Percentiles by Case and Source')
    ax.legend()
    plt.tight_layout()
    plt.show()
```

Combined Dataframe

Initial fitting

In [494…

```
df_name = df_combined
categorical_vars = ['job_category','seniority_level','country']
```

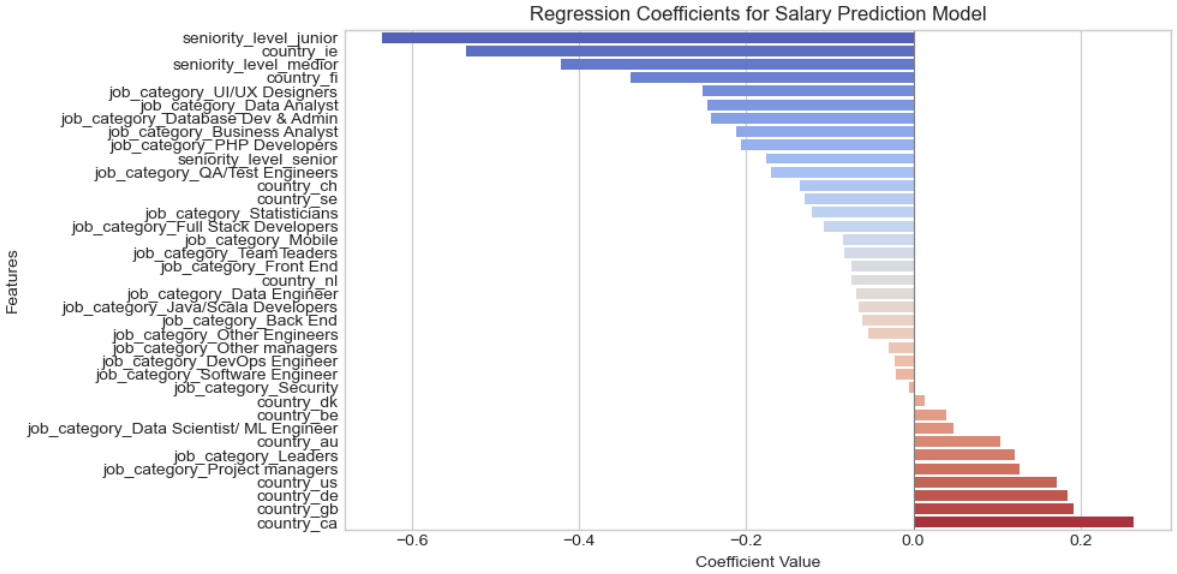
In [496…

```
# Your prepared DataFrame
X = df_name[categorical_vars]
y = df_name['salary_norm_2024_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)
model_for_coefficients_01 = model

# View the summary
print(model.summary())

# Print encoded feature categories (useful for future predictions)
print("\nEncoded feature categories:")
for feature, categories in encoded_features.items():
    print(f"{feature}: {categories}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     salary_norm_2024_log   R-squared:                       0.300
Model:                              OLS   Adj. R-squared:                  0.299
Method:                   Least Squares   F-statistic:                     308.1
Date:                  Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                          16:06:57   Log-Likelihood:                -10359.
No. Observations:                 26655   AIC:                         2.079e+04
Df Residuals:                     26617   BIC:                         2.110e+04
Df Model:                            37
Covariance Type:              nonrobust
==============================================================================================================
                                                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------
const                                          0.8191      0.056     14.512      0.000       0.708       0.930
job_category_Back End                         -0.0605      0.021     -2.851      0.004      -0.102      -0.019
job_category_Business Analyst                 -0.2122      0.021    -10.069      0.000      -0.254      -0.171
job_category_Data Analyst                     -0.2470      0.016    -15.616      0.000      -0.278      -0.216
job_category_Data Engineer                    -0.0689      0.016     -4.365      0.000      -0.100      -0.038
job_category_Data Scientist/ ML Engineer       0.0476      0.015      3.156      0.002       0.018       0.077
job_category_Database Dev & Admin             -0.2418      0.052     -4.625      0.000      -0.344      -0.139
job_category_DevOps Engineer                  -0.0229      0.028     -0.812      0.417      -0.078       0.032
job_category_Front End                        -0.0745      0.025     -2.929      0.003      -0.124      -0.025
job_category_Full Stack Developers            -0.1067      0.042     -2.562      0.010      -0.188      -0.025
job_category_Java/Scala Developers            -0.0660      0.044     -1.510      0.131      -0.152       0.020
job_category_Leaders                           0.1211      0.024      5.049      0.000       0.074       0.168
job_category_Mobile                           -0.0846      0.030     -2.793      0.005      -0.144      -0.025
job_category_Other Engineers                  -0.0544      0.080     -0.682      0.495      -0.211       0.102
job_category_Other managers                   -0.0297      0.021     -1.419      0.156      -0.071       0.011
job_category_PHP Developers                   -0.2056      0.081     -2.525      0.012      -0.365      -0.046
job_category_Project managers                  0.1269      0.019      6.636      0.000       0.089       0.164
job_category_QA/Test Engineers                -0.1707      0.027     -6.355      0.000      -0.223      -0.118
job_category_Security                         -0.0049      0.080     -0.062      0.951      -0.161       0.151
job_category_Software Engineer                -0.0216      0.017     -1.280      0.200      -0.055       0.011
job_category_Statisticians                    -0.1215      0.035     -3.478      0.001      -0.190      -0.053
job_category_Team leaders                     -0.0826      0.031     -2.632      0.008      -0.144      -0.021
job_category_UI/UX Designers                  -0.2527      0.068     -3.706      0.000      -0.386      -0.119
seniority_level_junior                        -0.6358      0.011    -59.000      0.000      -0.657      -0.615
seniority_level_medior                        -0.4222      0.009    -46.013      0.000      -0.440      -0.404
seniority_level_senior                        -0.1754      0.008    -20.718      0.000      -0.192      -0.159
country_au                                     0.1034      0.057      1.824      0.068      -0.008       0.215
country_be                                     0.0398      0.067      0.597      0.550      -0.091       0.170
country_ca                                     0.2632      0.055      4.769      0.000       0.155       0.371
country_ch                                    -0.1354      0.063     -2.145      0.032      -0.259      -0.012
country_de                                     0.1845      0.054      3.397      0.001       0.078       0.291
country_dk                                     0.0136      0.074      0.185      0.853      -0.130       0.158
country_fi                                    -0.3379      0.169     -2.004      0.045      -0.668      -0.007
country_gb                                     0.1921      0.055      3.510      0.000       0.085       0.299
country_ie                                    -0.5354      0.064     -8.420      0.000      -0.660      -0.411
country_nl                                    -0.0745      0.058     -1.279      0.201      -0.189       0.040
country_se                                    -0.1302      0.066     -1.966      0.049      -0.260      -0.000
country_us                                     0.1718      0.054      3.184      0.001       0.066       0.278
==============================================================================
Omnibus:                     1799.110   Durbin-Watson:                   1.968
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5773.580
Skew:                           0.321   Prob(JB):                         0.00
Kurtosis:                       5.188   Cond. No.                         141.
==============================================================================

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

Encoded feature categories:
job_category: ['Architects', 'Back End', 'Business Analyst', 'Data Analyst', 'Data Engineer', 'Data Scientist/ ML Engineer', 'Database Dev & Admin', 'DevOps Engineer', 'Front End', 'Full Stack Developers', 'Java/Scala Developers', 'Leaders', 'Mo
bile', 'Other Engineers', 'Other managers', 'PHP Developers', 'Project managers', 'QA/Test Engineers', 'Security', 'Software Engineer', 'Statisticians', 'Team leaders', 'UI/UX Designers']
seniority_level: ['executive', 'junior', 'medior', 'senior']
country: ['at', 'au', 'be', 'ca', 'ch', 'de', 'dk', 'fi', 'gb', 'ie', 'nl', 'se', 'us']
```

In [60]:
```
rmse_original_scale = calculate_rmse_original_scale(model, y_aligned)
print("RMSE on Original Scale:", rmse_original_scale)

plot_model_coefficients(model,10,5)
```

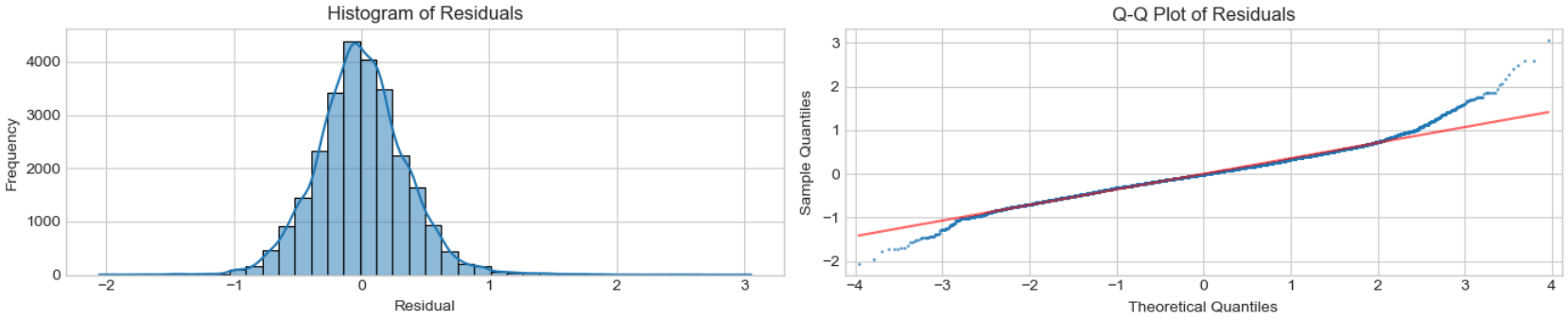
```
RMSE on Original Scale: 0.9835425668624351
```



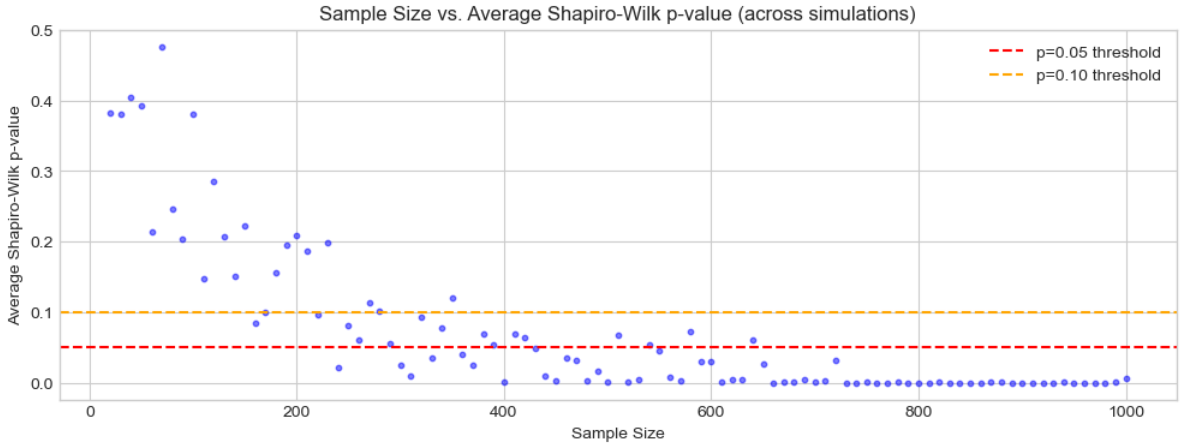
Checking Residuals: Normality

In [62]:
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [63]:
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```

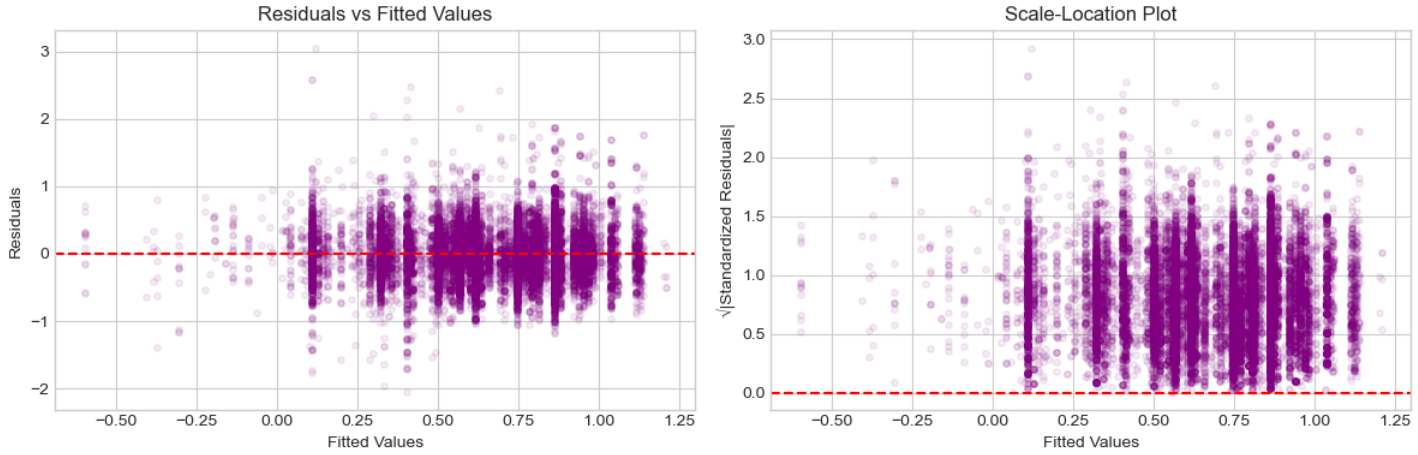


In [64]:
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```

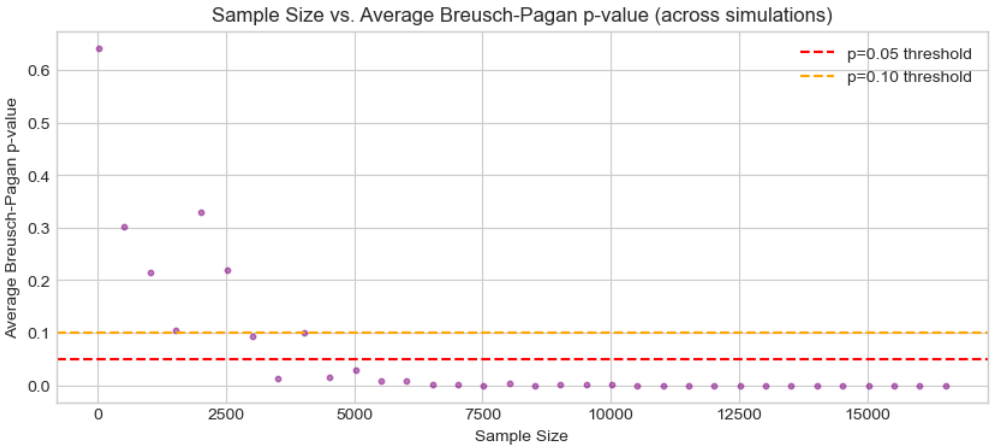


Checking Residuals: Homoscedasticity

In [66]:
```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```

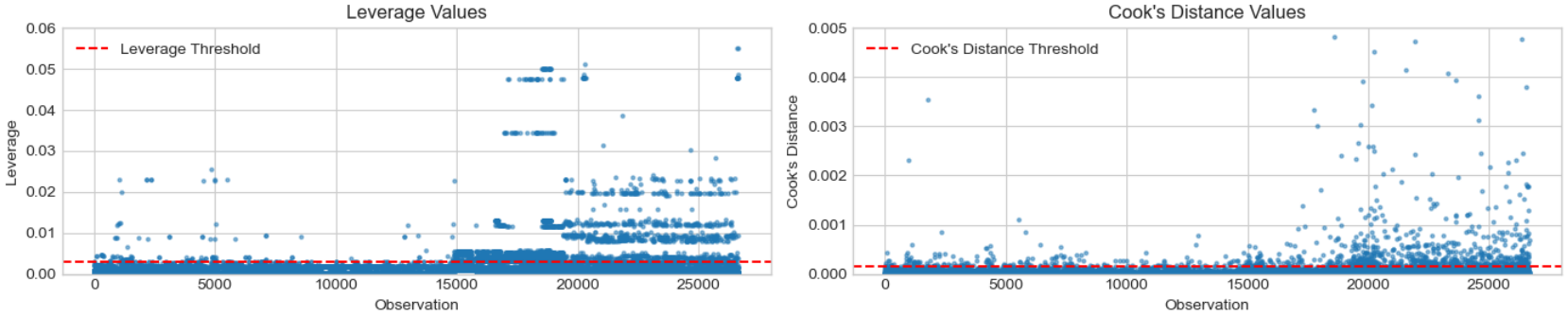


In [67]:
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=17000, step_size=500, num_simulations=10, dot_size=10)
```



Influential points

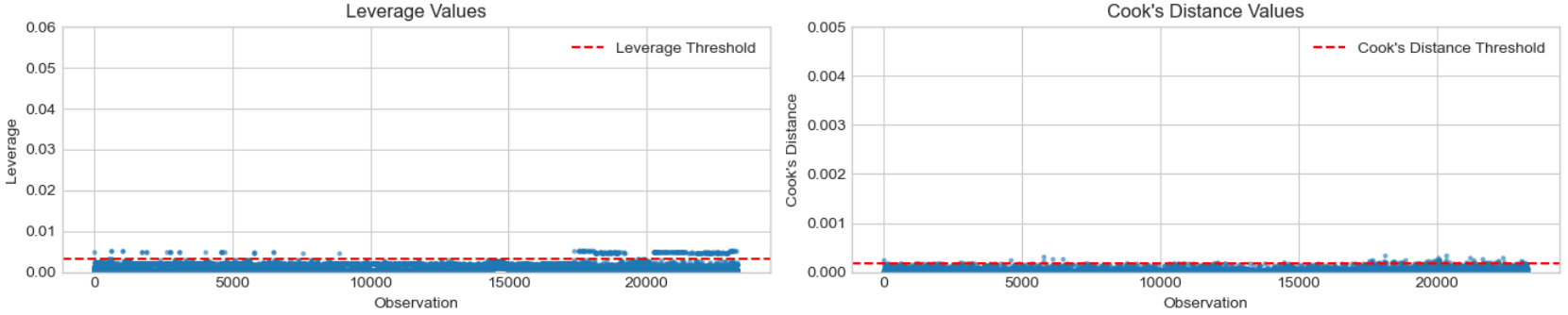
In [69]:
```
# Calculate Leverage and Cook's Distance using the model
leverage, cooks_d = calculate_leverage_and_cooks_distance(model)

# Plot the Leverage and Cook's Distance
plot_leverage_and_cooks_distance(leverage, cooks_d)
```



In [70]:
```
# Leverage OR cook's D EITHER high
# Identify observations with high leverage or high Cook's Distance
n = len(leverage)
p = model.df_model
leverage_threshold = 2 * (p + 1) / n
cooks_threshold = 4 / n

# Identify high leverage and high Cook's Distance points separately
high_leverage_points = np.where(leverage > leverage_threshold)[0]
high_cooks_points = np.where(cooks_d > cooks_threshold)[0]

# Use the union of high leverage and high Cook's Distance points
influential_points = np.union1d(high_leverage_points, high_cooks_points)

print(f"Number of high leverage points: {len(high_leverage_points)}")
print(f"Number of high Cook's Distance points: {len(high_cooks_points)}")
print(f"Number of influential points (either condition): {len(influential_points)}")

# Remove influential observations
X_encoded_cleaned = X_encoded.drop(index=influential_points).reset_index(drop=True)
y_cleaned = y_aligned.drop(index=influential_points).reset_index(drop=True)

# Refit the model without influential observations
```

```
model_cleaned = sm.OLS(y_cleaned, X_encoded_cleaned).fit()

# Recalculate leverage and Cook's Distance
leverage_cleaned, cooks_d_cleaned = calculate_leverage_and_cooks_distance(model_cleaned)

# Plot again
plot_leverage_and_cooks_distance(leverage_cleaned, cooks_d_cleaned)
```

```
Number of high leverage points: 2921
Number of high Cook's Distance points: 1011
Number of influential points (either condition): 3370
```



In [71]:
```
# Ensure indices are aligned with the original DataFrame
X_encoded_with_index = X_encoded.reset_index(drop=True)
y_aligned_with_index = y_aligned.reset_index(drop=True)

# Original indices (positions after resetting index)
original_indices = X_encoded_with_index.index

# Map influential points to positions in df_combined
# Reset index of df_combined to ensure alignment
df_name_reset = df_name.reset_index(drop=True)

# Ensure that df_combined_reset has the same number of rows as X_encoded_with_index
assert len(df_name_reset) == len(X_encoded_with_index), "Mismatch in number of rows"

# Retrieve influential observations from the original DataFrame
df_name_influential = df_name_reset.iloc[influential_points]

# Remove the influential points from the original data frame, creating a new df
df_name_noninfl = df_name_reset.drop(index=influential_points).reset_index(drop=True)

# Examine influential observations
df_name_influential.head()
```

Out[71]:

	seniority_level	level_1	year	employment_status	job_title	salary_in_currency	salary_currency	salary	country	remote_ratio	company_location	company_size	ratio	survey	company_size_category	country_code	median_income_2020_usd	mean_income_202
2	junior	7.00	2024	ft	business intelligence analyst	157400.00	usd	157400.00	us	100.00	us	m	1.00	ai	m	us	19306	
18	junior	109.00	2024	ft	business intelligence analyst	164500.00	usd	164500.00	us	0.00	us	m	1.00	ai	m	us	19306	
32	junior	200.00	2024	ft	business intelligence	176627.00	usd	176627.00	us	0.00	us	m	1.00	ai	m	us	19306	
93	junior	775.00	2024	ft	business intelligence analyst	234000.00	usd	234000.00	us	0.00	us	m	1.00	ai	m	us	19306	
105	junior	859.00	2024	ft	data engineer	47820.00	eur	53133.00	nl	0.00	nl	m	1.11	ai	m	nl	17154	

In [72]:
```
print("Original DataFrame shape:", df_name_reset.shape)
print("Number of influential points:", len(influential_points))
print("DataFrame without influential points shape:", df_name_noninfl.shape)
```

```
Original DataFrame shape: (26655, 63)
Number of influential points: 3370
DataFrame without influential points shape: (23285, 63)
```

Comparing VIFs

In [74]:

```python
# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[74]:

	VIF	Feature
0	665.70	Intercept
37	133.31	C(country)[T.us]
30	96.71	C(country)[T.de]
33	30.92	C(country)[T.gb]
28	20.42	C(country)[T.ca]
5	11.52	C(job_category)[T.Data Scientist/ ML Engineer]
26	10.20	C(country)[T.au]
3	7.06	C(job_category)[T.Data Analyst]
35	6.92	C(country)[T.nl]
4	6.50	C(job_category)[T.Data Engineer]
19	4.37	C(job_category)[T.Software Engineer]
29	3.67	C(country)[T.ch]
25	3.65	C(seniority_level)[T.senior]
34	3.54	C(country)[T.ie]
24	3.15	C(seniority_level)[T.medior]
36	2.95	C(country)[T.se]
27	2.88	C(country)[T.be]
16	2.42	C(job_category)[T.Project managers]
1	2.23	C(job_category)[T.Back End]
23	2.18	C(seniority_level)[T.junior]
31	2.16	C(country)[T.dk]
14	1.95	C(job_category)[T.Other managers]
2	1.92	C(job_category)[T.Business Analyst]
11	1.67	C(job_category)[T.Leaders]
8	1.63	C(job_category)[T.Front End]
17	1.53	C(job_category)[T.QA/Test Engineers]
7	1.45	C(job_category)[T.DevOps Engineer]
21	1.42	C(job_category)[T.Team leaders]
12	1.37	C(job_category)[T.Mobile]
20	1.22	C(job_category)[T.Statisticians]
9	1.17	C(job_category)[T.Full Stack Developers]
10	1.15	C(job_category)[T.Java/Scala Developers]
32	1.11	C(country)[T.fi]
6	1.09	C(job_category)[T.Database Dev & Admin]
22	1.06	C(job_category)[T.UI/UX Designers]
13	1.05	C(job_category)[T.Other Engineers]
18	1.04	C(job_category)[T.Security]
15	1.04	C(job_category)[T.PHP Developers]

In [75]:
```
# Define the dataframe to be used
df_name = df_name_noninfl

# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])
```

```
# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[75]:

	VIF	Feature
0	166.10	Intercept
15	20.47	C(country)[T.us]
13	15.38	C(country)[T.de]
5	11.02	C(job_category)[T.Data Scientist/ ML Engineer]
3	7.17	C(job_category)[T.Data Analyst]
4	6.66	C(job_category)[T.Data Engineer]
14	6.57	C(country)[T.gb]
12	4.51	C(country)[T.ca]
8	4.30	C(job_category)[T.Software Engineer]
11	3.97	C(seniority_level)[T.senior]
10	3.42	C(seniority_level)[T.medior]
1	2.31	C(job_category)[T.Back End]
9	2.29	C(seniority_level)[T.junior]
7	2.22	C(job_category)[T.Project managers]
6	1.92	C(job_category)[T.Other managers]
2	1.89	C(job_category)[T.Business Analyst]

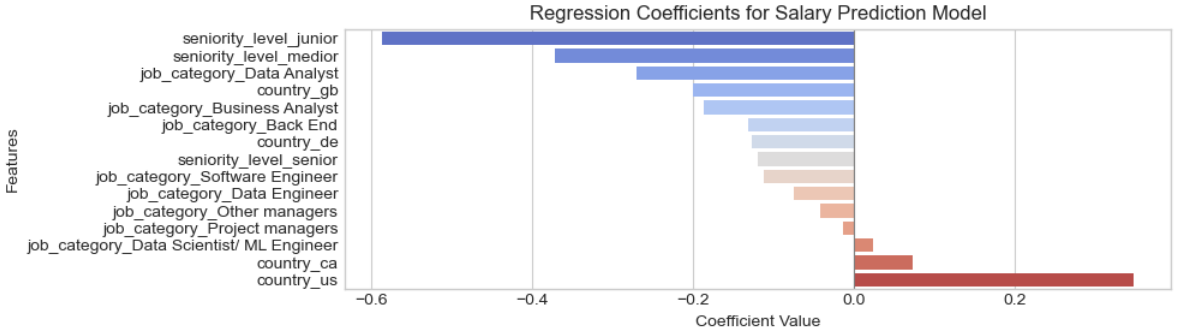
Refitting w/o influential points

In [77]:
```
# Your prepared DataFrame
X = df_name_noninfl[categorical_vars]
y = df_name_noninfl['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.444
Model:                            OLS   Adj. R-squared:                  0.443
Method:                 Least Squares   F-statistic:                     1237.
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        14:03:26   Log-Likelihood:                -8004.6
No. Observations:               23285   AIC:                         1.604e+04
Df Residuals:                   23269   BIC:                         1.617e+04
Df Model:                          15
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           11.7875      0.029    408.856      0.000      11.731      11.844
job_category_Back End           -0.1318      0.021     -6.357      0.000      -0.172      -0.091
job_category_Business Analyst   -0.1864      0.021     -8.852      0.000      -0.228      -0.145
job_category_Data Analyst       -0.2699      0.016    -17.240      0.000      -0.301      -0.239
job_category_Data Engineer      -0.0748      0.016     -4.791      0.000      -0.105      -0.044
job_category_Data Scientist/ ML Engineer     0.0239      0.015      1.598      0.110      -0.005       0.053
job_category_Other managers     -0.0415      0.021     -1.982      0.048      -0.083      -0.000
job_category_Project managers   -0.0135      0.020     -0.690      0.491      -0.052       0.025
job_category_Software Engineer  -0.1114      0.017     -6.630      0.000      -0.144      -0.078
seniority_level_junior          -0.5871      0.011    -51.605      0.000      -0.609      -0.565
seniority_level_medior          -0.3720      0.010    -38.136      0.000      -0.391      -0.353
seniority_level_senior          -0.1195      0.009    -13.158      0.000      -0.137      -0.102
country_ca                       0.0737      0.026      2.814      0.005       0.022       0.125
country_de                      -0.1274      0.024     -5.209      0.000      -0.175      -0.079
country_gb                      -0.1995      0.025     -7.957      0.000      -0.249      -0.150
country_us                       0.3479      0.023     14.892      0.000       0.302       0.394
==============================================================================
Omnibus:                      130.262   Durbin-Watson:                   1.925
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              169.288
Skew:                           0.090   Prob(JB):                     1.74e-37
Kurtosis:                       3.377   Cond. No.                         37.2
==============================================================================

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

In [78]:
```
rmse_original_scale = calculate_rmse_original_scale(model, y_aligned)
print("RMSE on Original Scale:", rmse_original_scale)

plot_model_coefficients(model,10,3)
```

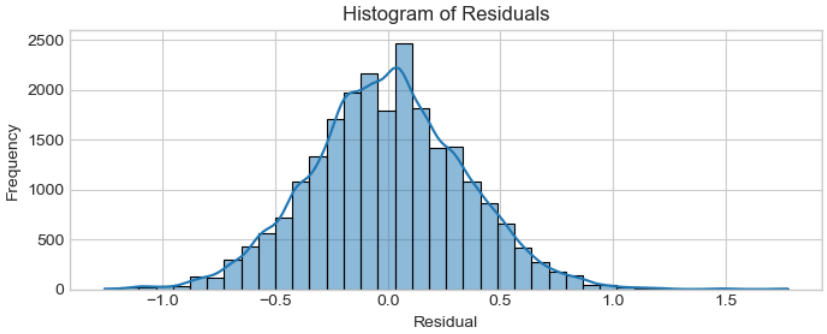
```
RMSE on Original Scale: 54169.48639308397
```

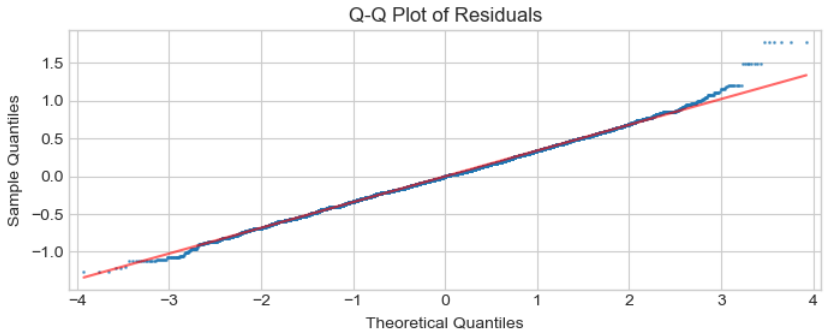


Checking Residuals: Normality

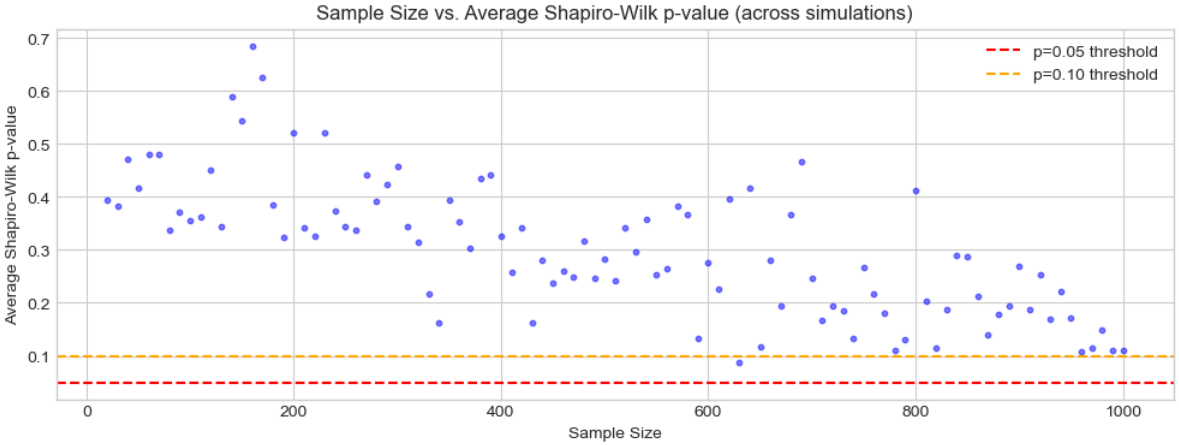
In [80]:
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [81]:
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



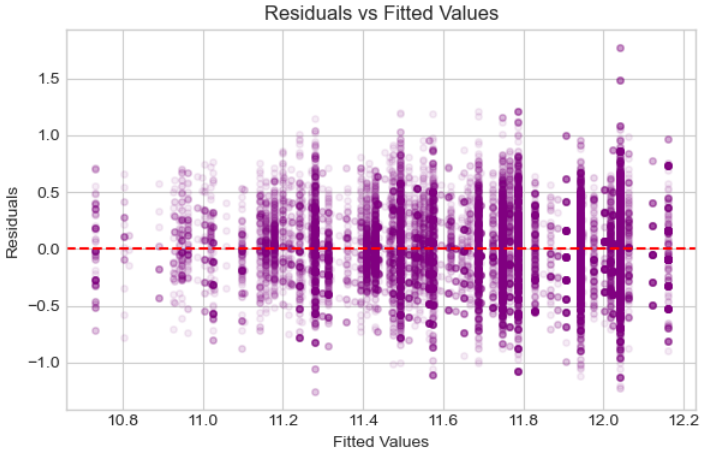


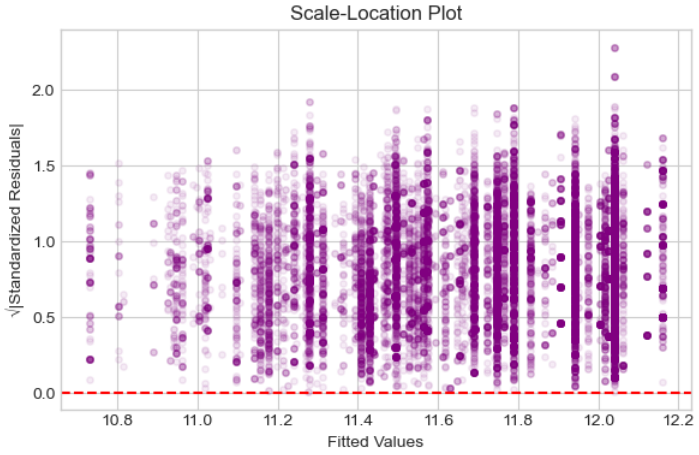
In [82]:
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



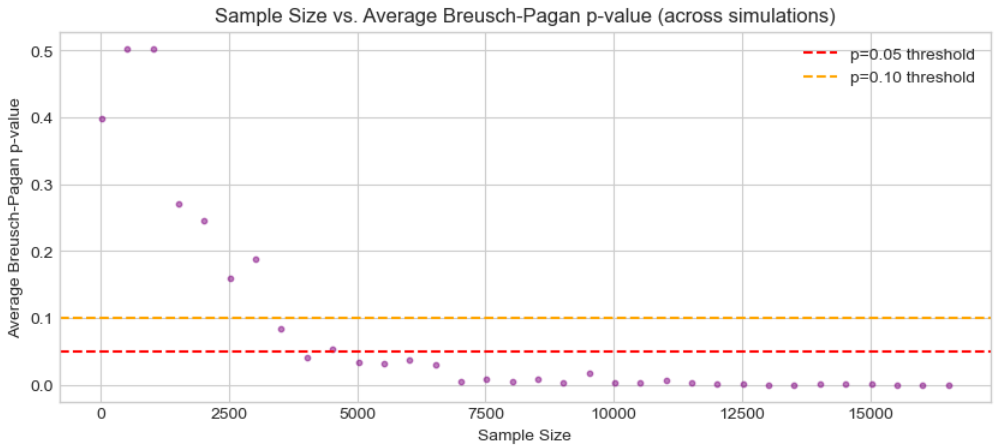
Checking Residuals: Homoscedasticity

In [84]:
```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```





In [85]:
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=17000, step_size=500, num_simulations=10, dot_size=10)
```



Making a prediction

In [87]:
```
specific_case = {
    'job_category': 'Data Scientist/ ML Engineer',
    'seniority_level': 'senior',
    'country': 'us'
}
```

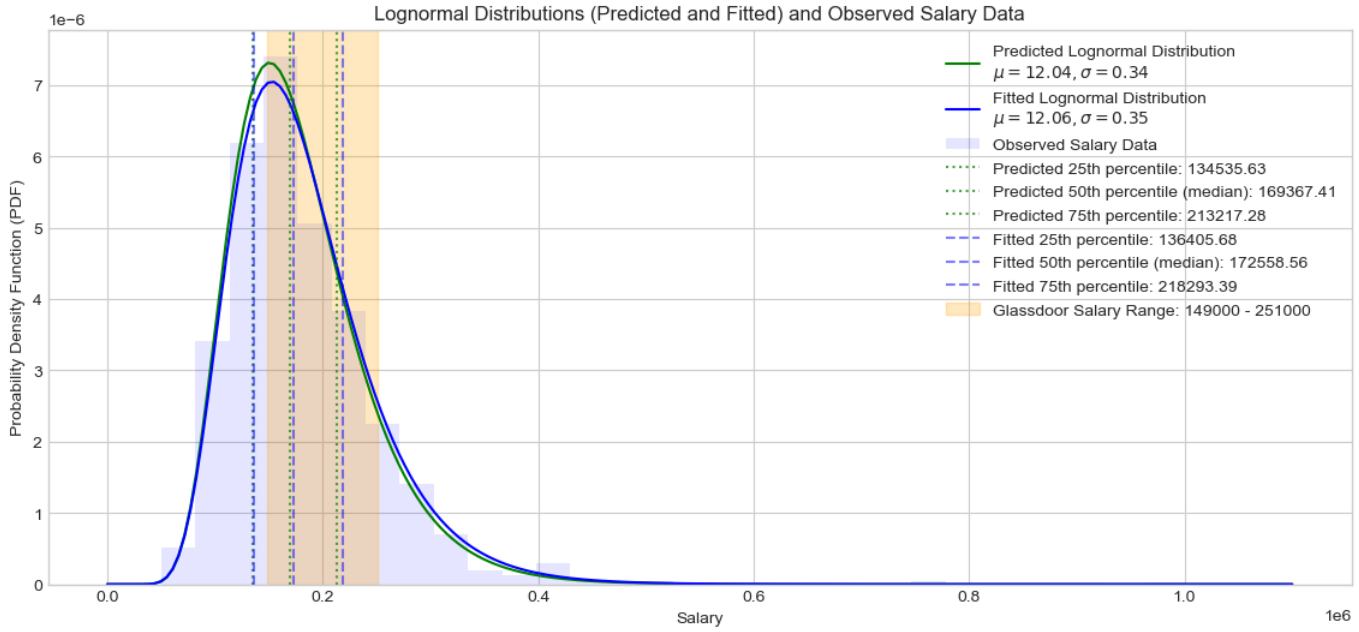
In [88]:
```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 169367.41
Predicted 25th Percentile Salary: 134535.63
Predicted 75th Percentile Salary: 213217.28
```

Plotting the prediction

In [90]:
```
specific_case = {
    'job_category': 'Data Scientist/ ML Engineer',
    'seniority_level': 'senior',
    'country': 'us'
}
```

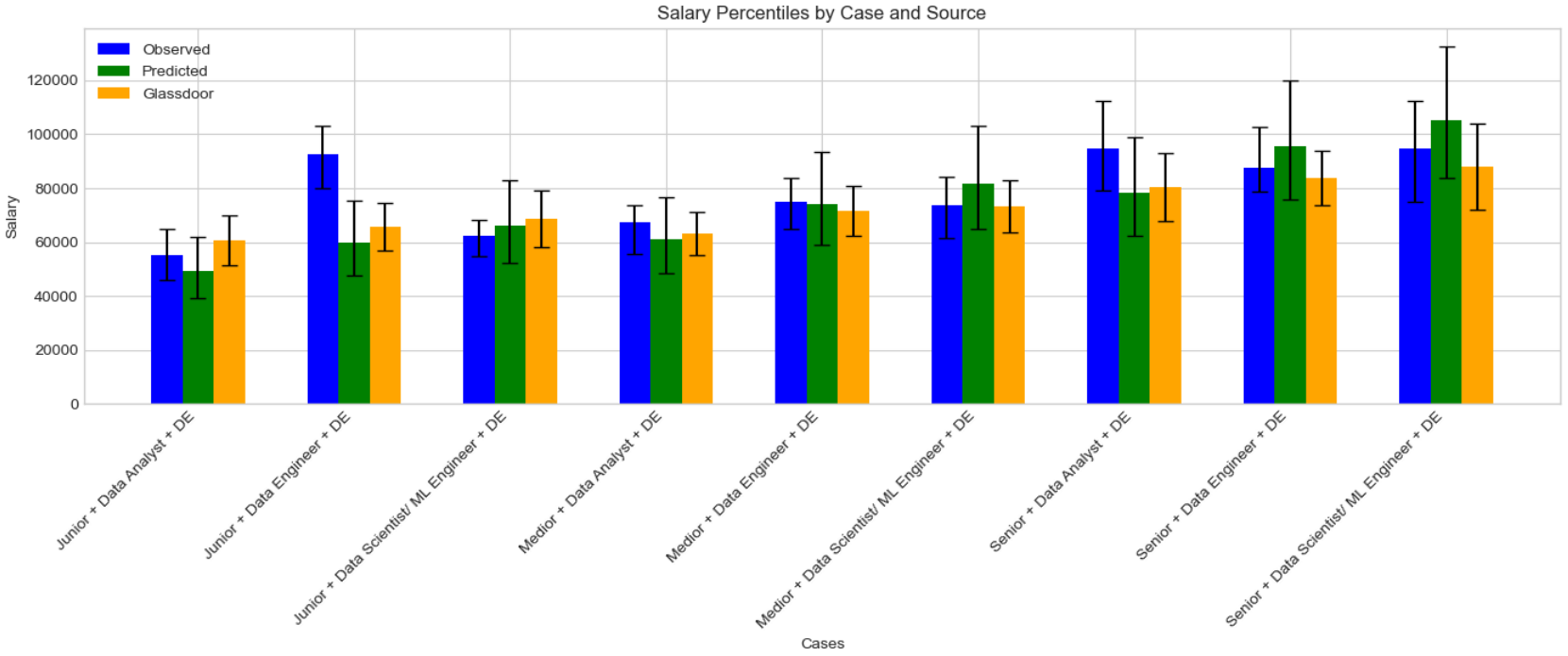
In [91]:
```
plot_salary_distributions(df_name, model, encoded_features, specific_case, glassdoor_data)
```



The result

In [93]:
```
# Define your specific cases in the desired order
specific_cases = [
    {'job_category': 'Data Analyst', 'seniority_level': 'junior', 'country': 'de'},
    {'job_category': 'Data Analyst', 'seniority_level': 'medior', 'country': 'de'},
    {'job_category': 'Data Analyst', 'seniority_level': 'senior', 'country': 'de'},
    {'job_category': 'Data Engineer', 'seniority_level': 'junior', 'country': 'de'},
    {'job_category': 'Data Engineer', 'seniority_level': 'medior', 'country': 'de'},
    {'job_category': 'Data Engineer', 'seniority_level': 'senior', 'country': 'de'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'junior', 'country': 'de'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'medior', 'country': 'de'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'senior', 'country': 'de'},
]
```

In [94]:
```
plot_salary_percentiles_by_case(df_name, model, encoded_features, specific_cases, glassdoor_data)
```



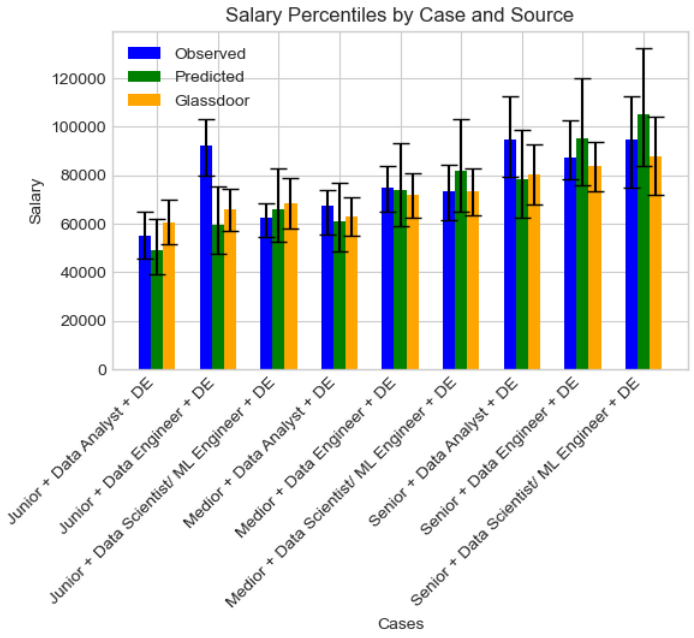
In [95]:

```python
def plot_salary_percentiles_by_case_2(df_name, model, encoded_features, specific_cases, glassdoor_data):
    """
    Plots salary percentiles (P25, P50, P75) for different cases and data sources (Observed, Predicted, Glassdoor).

    Parameters:
        df_name (pd.DataFrame): DataFrame containing observed salary data.
        model: Model for predicting salary.
        encoded_features: Encoded features for prediction.
        specific_cases (list of dict): List of specific cases to analyze, with keys 'job_category', 'seniority_level', and 'country'.
        glassdoor_data (pd.DataFrame): DataFrame with Glassdoor salary range data.

    Returns:
        None
    """
    # Initialize lists to store data
    case_labels = []
    observed_percentiles = []
    predicted_percentiles = []
    glassdoor_percentiles = []

    for specific_case in specific_cases:
        job_category = specific_case['job_category']
        seniority_level = specific_case['seniority_level']
        country = specific_case['country']

        # --- Observed Data ---
        salary_data = df_name['salary'][
            (df_name['seniority_level'] == seniority_level) &
            (df_name['job_category'] == job_category) &
            (df_name['country'] == country)
        ]

        if len(salary_data) > 0:
            P25_obs = np.percentile(salary_data, 25)
            P50_obs = np.percentile(salary_data, 50)
            P75_obs = np.percentile(salary_data, 75)
        else:
            P25_obs = P50_obs = P75_obs = np.nan

        observed_percentiles.append([P25_obs, P50_obs, P75_obs])

        # --- Predicted Data ---
```

```
    predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(
        model, encoded_features, specific_case
    )

    P25_pred, P50_pred, P75_pred = predicted_salary_p25, predicted_salary, predicted_salary_p75
    predicted_percentiles.append([P25_pred, P50_pred, P75_pred])

    # --- Glassdoor Data ---
    glassdoor_row = glassdoor_data[
        (glassdoor_data['job_category'] == job_category) &
        (glassdoor_data['seniority_level'] == seniority_level) &
        (glassdoor_data['country'] == country)
    ]

    if not glassdoor_row.empty:
        glassdoor_lower = glassdoor_row['glassdoor_lower'].values[0]
        glassdoor_upper = glassdoor_row['glassdoor_upper'].values[0]
        glassdoor_median = (glassdoor_lower + glassdoor_upper) / 2
        P25_glassdoor, P50_glassdoor, P75_glassdoor = glassdoor_lower, glassdoor_median, glassdoor_upper
    else:
        P25_glassdoor = P50_glassdoor = P75_glassdoor = np.nan

    glassdoor_percentiles.append([P25_glassdoor, P50_glassdoor, P75_glassdoor])

    # Add case label
    case_label = f"{seniority_level.capitalize()} + {job_category} + {country.upper()}"
    case_labels.append(case_label)

# Create DataFrame for plotting
data_list = []
for idx, case_label in enumerate(case_labels):
    data_list.append({'Case': case_label, 'Source': 'Observed', 'P25': observed_percentiles[idx][0], 'P50': observed_percentiles[idx][1], 'P75': observed_percentiles[idx][2]})
    data_list.append({'Case': case_label, 'Source': 'Predicted', 'P25': predicted_percentiles[idx][0], 'P50': predicted_percentiles[idx][1], 'P75': predicted_percentiles[idx][2]})
    data_list.append({'Case': case_label, 'Source': 'Glassdoor', 'P25': glassdoor_percentiles[idx][0], 'P50': glassdoor_percentiles[idx][1], 'P75': glassdoor_percentiles[idx][2]})

df_plot = pd.DataFrame(data_list)
df_plot.sort_values(['Case', 'Source'], inplace=True)

# Plotting
cases = df_plot['Case'].unique()
sources = ['Observed', 'Predicted', 'Glassdoor']
n_cases = len(cases)
n_sources = len(sources)
bar_width = 0.2
x = np.arange(n_cases)
offsets = np.linspace(-bar_width * (n_sources - 1) / 2, bar_width * (n_sources - 1) / 2, n_sources)
source_colors = {'Observed': 'blue', 'Predicted': 'green', 'Glassdoor': 'orange'}

fig, ax = plt.subplots(figsize=(6, 5.5))
plotted_labels = set()

for i, source in enumerate(sources):
    bar_positions = x + offsets[i]
    medians, lower_errors, upper_errors = [], [], []

    for case in cases:
        source_data = df_plot[(df_plot['Case'] == case) & (df_plot['Source'] == source)]
        if not source_data.empty:
            median = source_data['P50'].values[0]
            P25 = source_data['P25'].values[0]
            P75 = source_data['P75'].values[0]
            lower_error = median - P25
            upper_error = P75 - median
        else:
            median = np.nan
            lower_error = upper_error = np.nan
        medians.append(median)
        lower_errors.append(lower_error)
        upper_errors.append(upper_error)

    # Convert to numpy arrays and plot
    medians, lower_errors, upper_errors = np.array(medians), np.array(lower_errors), np.array(upper_errors)
    valid = ~np.isnan(medians)

    if valid.any():
        if source not in plotted_labels:
            ax.bar(bar_positions[valid], medians[valid], width=bar_width, label=source,
```

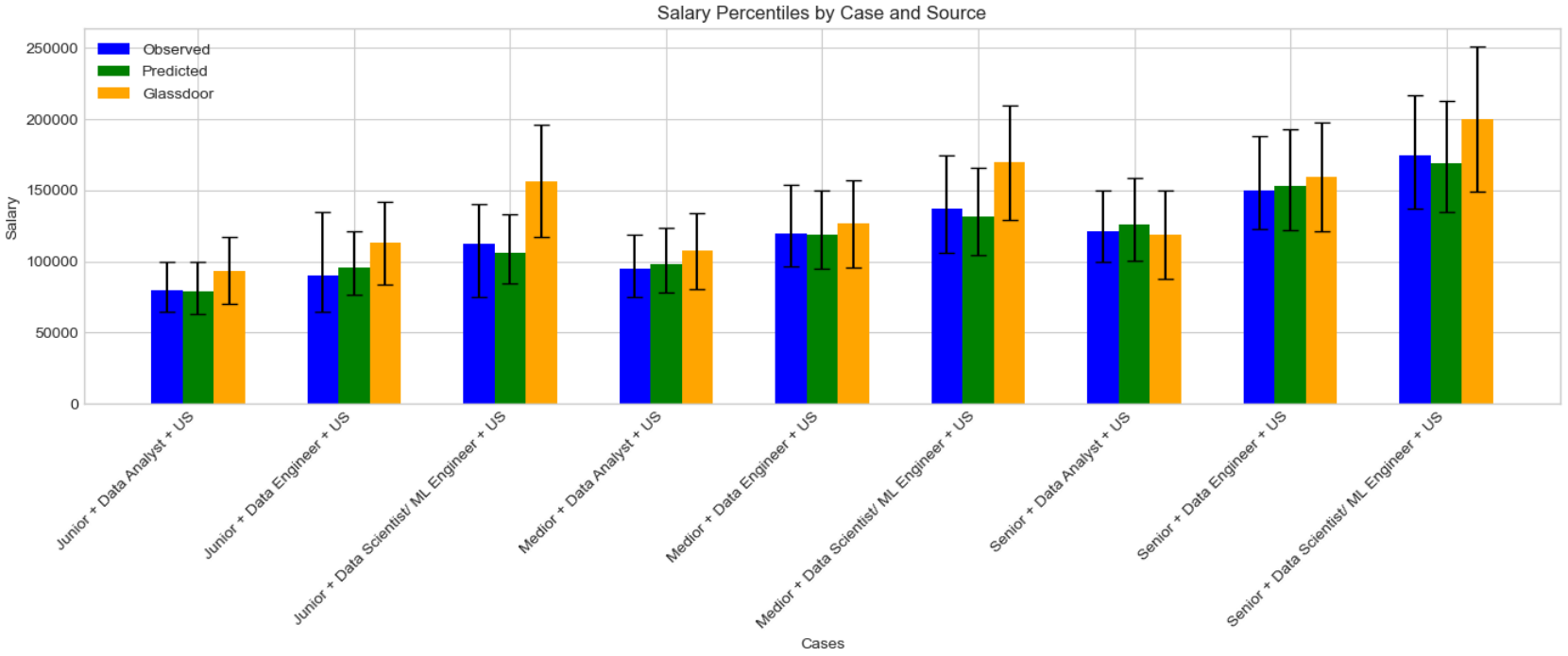
```
                    color=source_colors.get(source), yerr=[lower_errors[valid], upper_errors[valid]], capsize=5)
                plotted_labels.add(source)
            else:
                ax.bar(bar_positions[valid], medians[valid], width=bar_width,
                    color=source_colors.get(source), yerr=[lower_errors[valid], upper_errors[valid]], capsize=5)

    ax.set_xticks(x)
    ax.set_xticklabels(cases, rotation=45, ha='right')
    ax.set_xlabel('Cases')
    ax.set_ylabel('Salary')
    ax.set_title('Salary Percentiles by Case and Source')
    ax.legend()
    plt.tight_layout()
    plt.show()
```

In [96]:
```
plot_salary_percentiles_by_case_2(df_name, model, encoded_features, specific_cases, glassdoor_data)
```



In [97]:
```
# Define your specific cases in the desired order
specific_cases = [
    {'job_category': 'Data Analyst', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Analyst', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Analyst', 'seniority_level': 'senior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'senior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'senior', 'country': 'us'},
]
```

In [98]:
```
plot_salary_percentiles_by_case(df_name, model, encoded_features, specific_cases, glassdoor_data)
```



DF-AI

Initial fitting

In [101…
```
df_name = df_ai_w_l
categorical_vars = ['job_category','seniority_level','country']
```

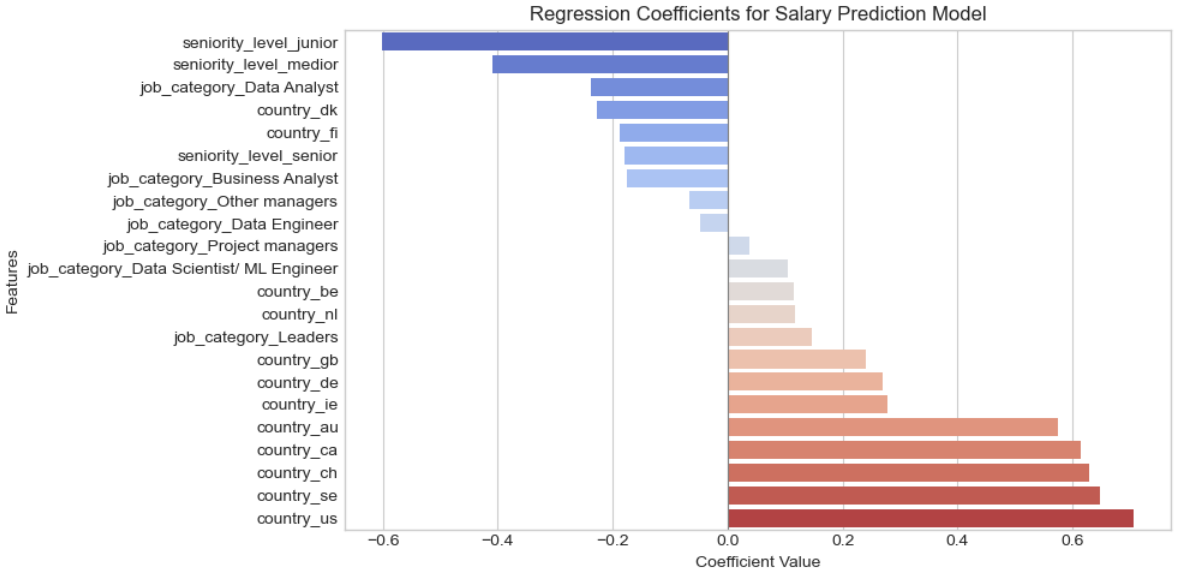
In [102…
```
# Your prepared DataFrame
X = df_name[categorical_vars]
y = df_name['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())

# Print encoded feature categories (useful for future predictions)
print("\nEncoded feature categories:")
for feature, categories in encoded_features.items():
    print(f"{feature}: {categories}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.351
Model:                            OLS   Adj. R-squared:                  0.350
Method:                 Least Squares   F-statistic:                     364.7
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        14:03:33   Log-Likelihood:                -5171.8
No. Observations:               14883   AIC:                         1.039e+04
Df Residuals:                   14860   BIC:                         1.056e+04
Df Model:                          22
Covariance Type:            nonrobust
================================================================================================
                                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           11.4574      0.106    107.722      0.000      11.249      11.666
job_category_Business Analyst   -0.1759      0.021     -8.207      0.000      -0.218      -0.134
job_category_Data Analyst       -0.2372      0.017    -13.885      0.000      -0.271      -0.204
job_category_Data Engineer      -0.0481      0.017     -2.857      0.004      -0.081      -0.015
job_category_Data Scientist/ ML Engineer     0.1044      0.016      6.404      0.000       0.072       0.136
job_category_Leaders             0.1465      0.030      4.951      0.000       0.089       0.205
job_category_Other managers     -0.0662      0.024     -2.795      0.005      -0.113      -0.020
job_category_Project managers    0.0385      0.078      0.492      0.623      -0.115       0.192
seniority_level_junior          -0.6011      0.021    -29.180      0.000      -0.641      -0.561
seniority_level_medior          -0.4095      0.018    -22.209      0.000      -0.446      -0.373
seniority_level_senior          -0.1798      0.018    -10.095      0.000      -0.215      -0.145
country_au                       0.5751      0.114      5.025      0.000       0.351       0.799
country_be                       0.1147      0.200      0.573      0.567      -0.278       0.507
country_ca                       0.6150      0.105      5.867      0.000       0.410       0.820
country_ch                       0.6287      0.159      3.945      0.000       0.316       0.941
country_de                       0.2695      0.110      2.454      0.014       0.054       0.485
country_dk                      -0.2266      0.358     -0.633      0.527      -0.929       0.476
country_fi                      -0.1874      0.185     -1.013      0.311      -0.550       0.175
country_gb                       0.2412      0.104      2.313      0.021       0.037       0.446
country_ie                       0.2782      0.143      1.943      0.052      -0.002       0.559
country_nl                       0.1174      0.123      0.957      0.339      -0.123       0.358
country_se                       0.6481      0.264      2.457      0.014       0.131       1.165
country_us                       0.7058      0.104      6.818      0.000       0.503       0.909
==============================================================================
Omnibus:                        2.062   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.357   Jarque-Bera (JB):                2.048
Skew:                          -0.028   Prob(JB):                        0.359
Kurtosis:                       3.008   Cond. No.                         244.
==============================================================================

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

Encoded feature categories:
job_category: ['Architects', 'Business Analyst', 'Data Analyst', 'Data Engineer', 'Data Scientist/ ML Engineer', 'Leaders', 'Other managers', 'Project managers']
seniority_level: ['executive', 'junior', 'medior', 'senior']
country: ['at', 'au', 'be', 'ca', 'ch', 'de', 'dk', 'fi', 'gb', 'ie', 'nl', 'se', 'us']
```

In [103…

```python
rmse_original_scale = calculate_rmse_original_scale(model, y_aligned)
print("RMSE on Original Scale:", rmse_original_scale)

plot_model_coefficients(model,10,5)
```

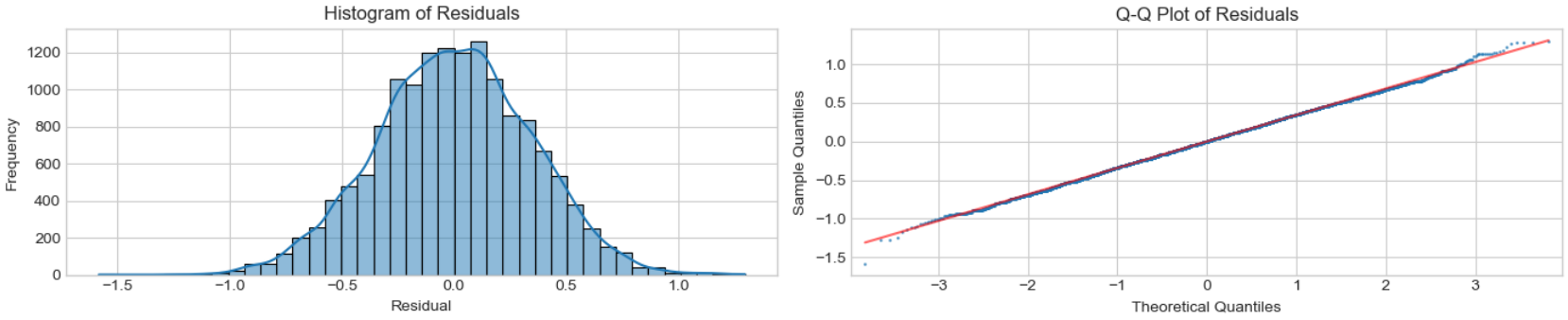
```
RMSE on Original Scale: 53773.94036886673
```



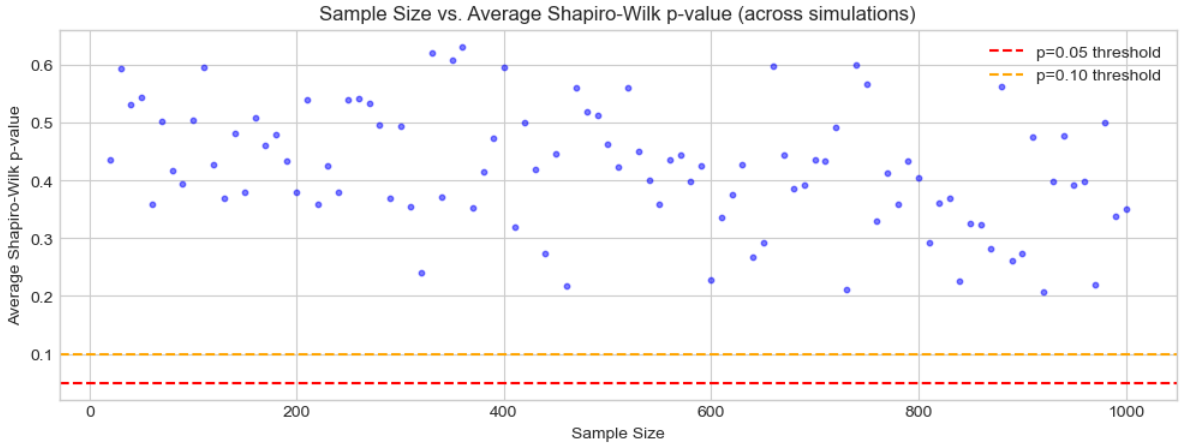
Checking Residuals: Normality

In [105…
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [106…
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



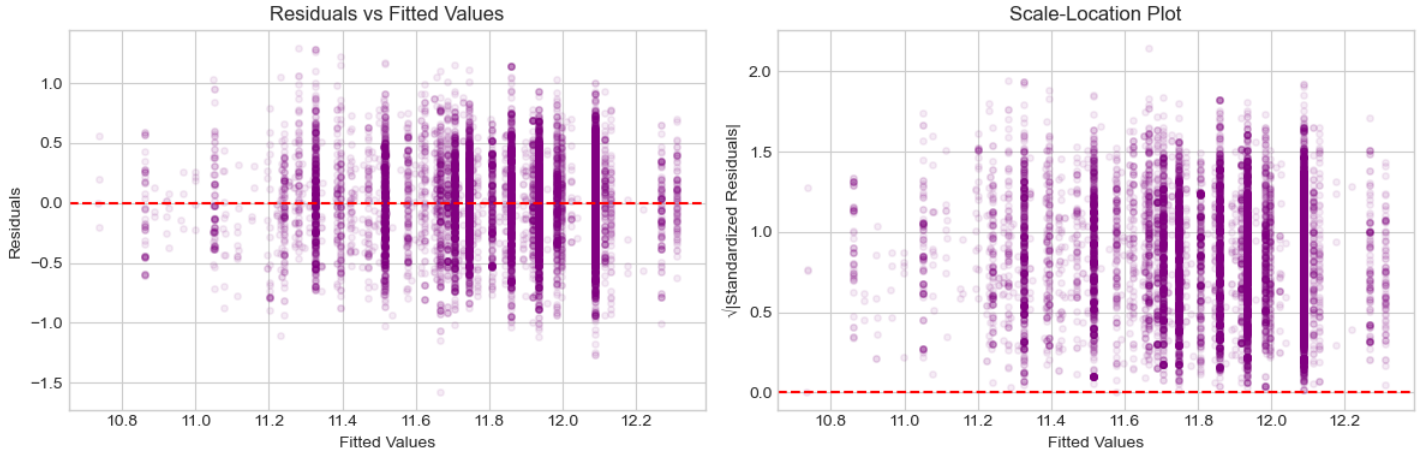
In [107…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



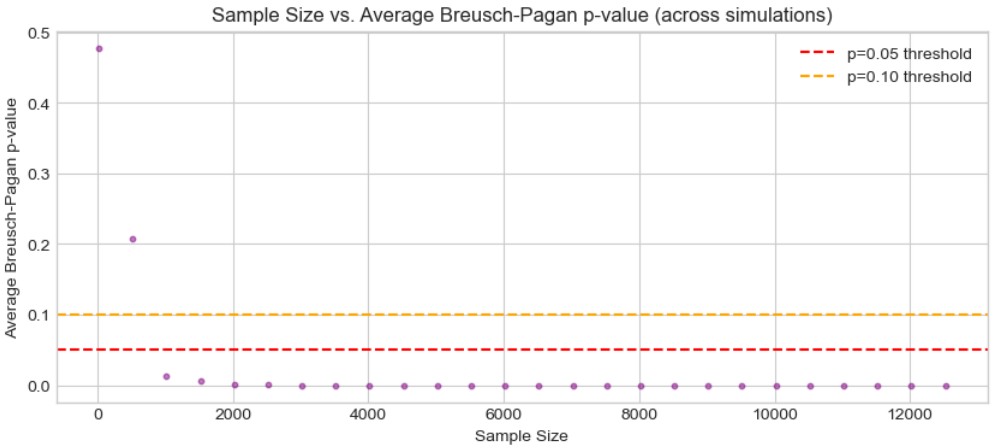
Checking Residuals: Homoscedasticity

In [109…

```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```



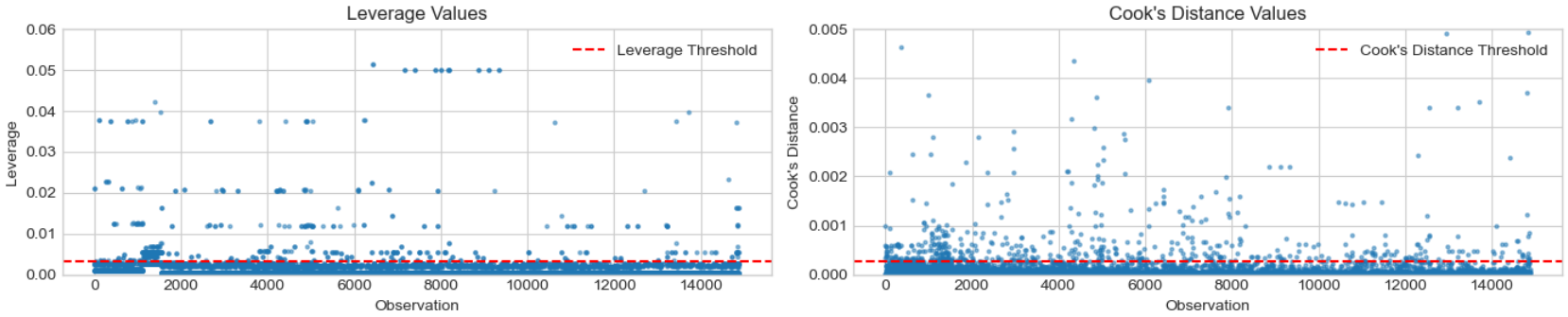
In [110…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=13000, step_size=500, num_simulations=10, dot_size=10)
```



Influential points

In [112…
```
# Calculate Leverage and Cook's Distance using the model
leverage, cooks_d = calculate_leverage_and_cooks_distance(model)

# Plot the Leverage and Cook's Distance
plot_leverage_and_cooks_distance(leverage, cooks_d)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\statsmodels\stats\outliers_influence.py:847: RuntimeWarning: invalid value encountered in sqrt
  return self.resid / sigma / np.sqrt(1 - hii)
```

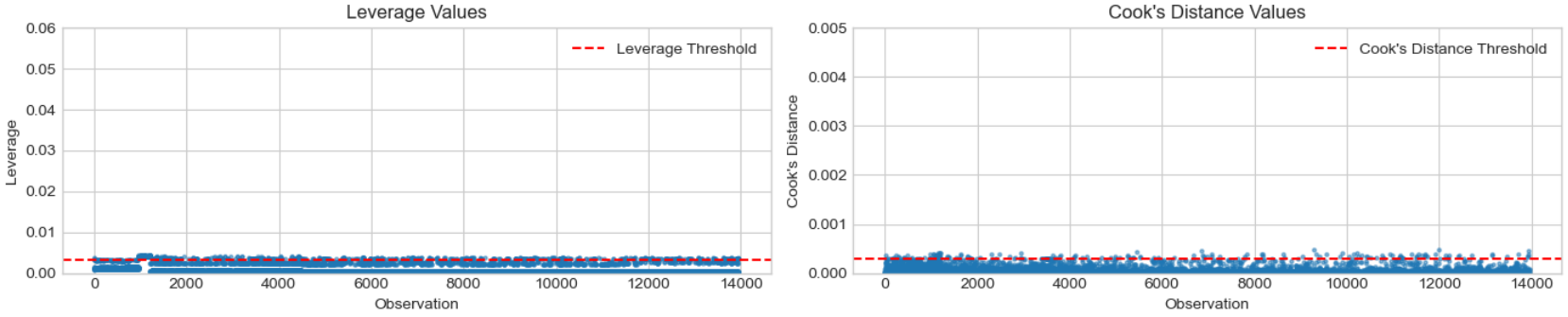


Note: The "invalid value encountered in sqrt" warning arises due to high-leverage points with leverage values close to 1, which lead to near-zero denominators in the Cook's Distance calculation. This can indicate influential points, which we consider informative in understanding the data distribution and model influence.

In [114…
```
# Leverage OR cook's D EITHER high
# Identify observations with high leverage or high Cook's Distance
n = len(leverage)
p = model.df_model
leverage_threshold = 2 * (p + 1) / n
cooks_threshold = 4 / n

# Identify high leverage and high Cook's Distance points separately
high_leverage_points = np.where(leverage > leverage_threshold)[0]
high_cooks_points = np.where(cooks_d > cooks_threshold)[0]

# Use the union of high leverage and high Cook's Distance points
influential_points = np.union1d(high_leverage_points, high_cooks_points)

print(f"Number of high leverage points: {len(high_leverage_points)}")
print(f"Number of high Cook's Distance points: {len(high_cooks_points)}")
print(f"Number of influential points (either condition): {len(influential_points)}")
```

```
# Remove influential observations
X_encoded_cleaned = X_encoded.drop(index=influential_points).reset_index(drop=True)
y_cleaned = y_aligned.drop(index=influential_points).reset_index(drop=True)

# Refit the model without influential observations
model_cleaned = sm.OLS(y_cleaned, X_encoded_cleaned).fit()

# Recalculate leverage and Cook's Distance
leverage_cleaned, cooks_d_cleaned = calculate_leverage_and_cooks_distance(model_cleaned)

# Plot again
plot_leverage_and_cooks_distance(leverage_cleaned, cooks_d_cleaned)
```

```
Number of high leverage points: 596
Number of high Cook's Distance points: 580
Number of influential points (either condition): 936
```



In [115…

```
# Ensure indices are aligned with the original DataFrame
X_encoded_with_index = X_encoded.reset_index(drop=True)
y_aligned_with_index = y_aligned.reset_index(drop=True)

# Original indices (positions after resetting index)
original_indices = X_encoded_with_index.index

# Map influential points to positions in df_combined
# Reset index of df_combined to ensure alignment
df_name_reset = df_name.reset_index(drop=True)

# Ensure that df_combined_reset has the same number of rows as X_encoded_with_index
assert len(df_name_reset) == len(X_encoded_with_index), "Mismatch in number of rows"

# Retrieve influential observations from the original DataFrame
df_name_influential = df_name_reset.iloc[influential_points]

# Remove the influential points from the original data frame, creating a new df
df_name_noninfl = df_name_reset.drop(index=influential_points).reset_index(drop=True)

# Examine influential observations
df_name_influential.head()
```

Out[115…

	seniority_level	level_1	year	employment_status	job_title	salary_in_currency	salary_currency	salary	country	remote_ratio	company_location	company_size	ratio	survey	company_size_category	country_code	median_income_2020_usd	mean_income_2020_u
0	junior	5	2024	ft	data analyst	102927	usd	102927	au	0	au	m	1.00	ai	m	au	17076	213
1	junior	6	2024	ft	data analyst	95010	usd	95010	au	0	au	m	1.00	ai	m	au	17076	213
2	junior	7	2024	ft	business intelligence analyst	157400	usd	157400	us	100	us	m	1.00	ai	m	us	19306	253
11	junior	88	2024	ft	data scientist	30000	gbp	37500	gb	0	gb	m	1.25	ai	m	gb	14793	181
16	junior	105	2024	ft	data scientist	26800	gbp	33500	gb	0	gb	m	1.25	ai	m	gb	14793	181

In [116…
```
print("Original DataFrame shape:", df_name_reset.shape)
print("Number of influential points:", len(influential_points))
print("DataFrame without influential points shape:", df_name_noninfl.shape)
```

```
Original DataFrame shape: (14883, 43)
Number of influential points: 936
DataFrame without influential points shape: (13947, 43)
```

Comparing VIFs

In [118…
```
# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[118…

	VIF	Feature
0	1432.94	Intercept
22	106.68	C(country)[T.us]
18	57.95	C(country)[T.gb]
13	38.16	C(country)[T.ca]
10	9.00	C(seniority_level)[T.senior]
15	8.78	C(country)[T.de]
4	8.32	C(job_category)[T.Data Scientist/ ML Engineer]
9	7.74	C(seniority_level)[T.medior]
3	6.38	C(job_category)[T.Data Engineer]
2	6.12	C(job_category)[T.Data Analyst]
11	5.44	C(country)[T.au]
8	3.72	C(seniority_level)[T.junior]
20	3.45	C(country)[T.nl]
1	2.12	C(job_category)[T.Business Analyst]
19	2.09	C(country)[T.ie]
6	1.79	C(job_category)[T.Other managers]
14	1.73	C(country)[T.ch]
5	1.57	C(job_category)[T.Leaders]
17	1.46	C(country)[T.fi]
12	1.36	C(country)[T.be]
21	1.18	C(country)[T.se]
16	1.09	C(country)[T.dk]
7	1.04	C(job_category)[T.Project managers]

In [119…
```
# Define the dataframe to be used
df_name = df_name_noninfl

# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')
```

```
# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[119…

	VIF	Feature
0	146.00	Intercept
8	12.46	C(seniority_level)[T.senior]
7	10.78	C(seniority_level)[T.medior]
4	9.39	C(job_category)[T.Data Scientist/ ML Engineer]
3	7.28	C(job_category)[T.Data Engineer]
2	6.96	C(job_category)[T.Data Analyst]
6	4.60	C(seniority_level)[T.junior]
9	2.68	C(country)[T.gb]
10	2.65	C(country)[T.us]
1	2.13	C(job_category)[T.Business Analyst]
5	1.76	C(job_category)[T.Other managers]

Refitting w/o influential points

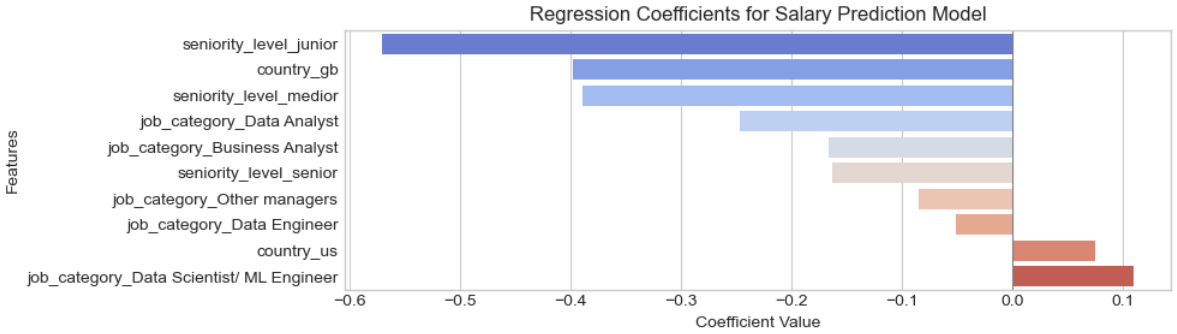
In [121…

```
# Your prepared DataFrame
X = df_name_noninfl[categorical_vars]
y = df_name_noninfl['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.352
Model:                            OLS   Adj. R-squared:                  0.351
Method:                 Least Squares   F-statistic:                     755.4
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        14:03:38   Log-Likelihood:                -4134.2
No. Observations:               13947   AIC:                             8290.
Df Residuals:                   13936   BIC:                             8373.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================================================
                                                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------
const                                         12.0708      0.033    362.356      0.000      12.005      12.136
job_category_Business Analyst                 -0.1669      0.022     -7.474      0.000      -0.211      -0.123
job_category_Data Analyst                     -0.2472      0.018    -13.988      0.000      -0.282      -0.213
job_category_Data Engineer                    -0.0509      0.017     -2.918      0.004      -0.085      -0.017
job_category_Data Scientist/ ML Engineer       0.1091      0.017      6.438      0.000       0.076       0.142
job_category_Other managers                   -0.0846      0.025     -3.386      0.001      -0.134      -0.036
seniority_level_junior                        -0.5708      0.023    -24.441      0.000      -0.617      -0.525
seniority_level_medior                        -0.3894      0.021    -18.219      0.000      -0.431      -0.347
seniority_level_senior                        -0.1636      0.021     -7.853      0.000      -0.204      -0.123
country_gb                                    -0.3982      0.025    -16.140      0.000      -0.447      -0.350
country_us                                     0.0744      0.020      3.775      0.000       0.036       0.113
==============================================================================
Omnibus:                       21.218   Durbin-Watson:                   2.113
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.256
Skew:                          -0.071   Prob(JB):                     3.99e-05
Kurtosis:                       2.878   Cond. No.                         29.4
==============================================================================

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

In [122…]
```
rmse_original_scale = calculate_rmse_original_scale(model, y_aligned)
print("RMSE on Original Scale:", rmse_original_scale)

plot_model_coefficients(model,10,3)
```

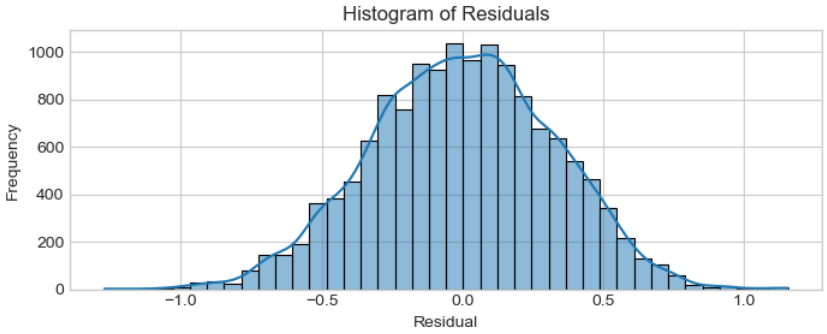
```
RMSE on Original Scale: 51446.312660588315
```

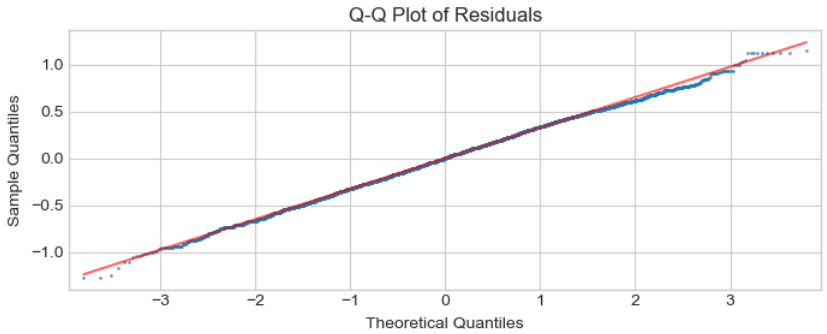


Checking Residuals: Normality

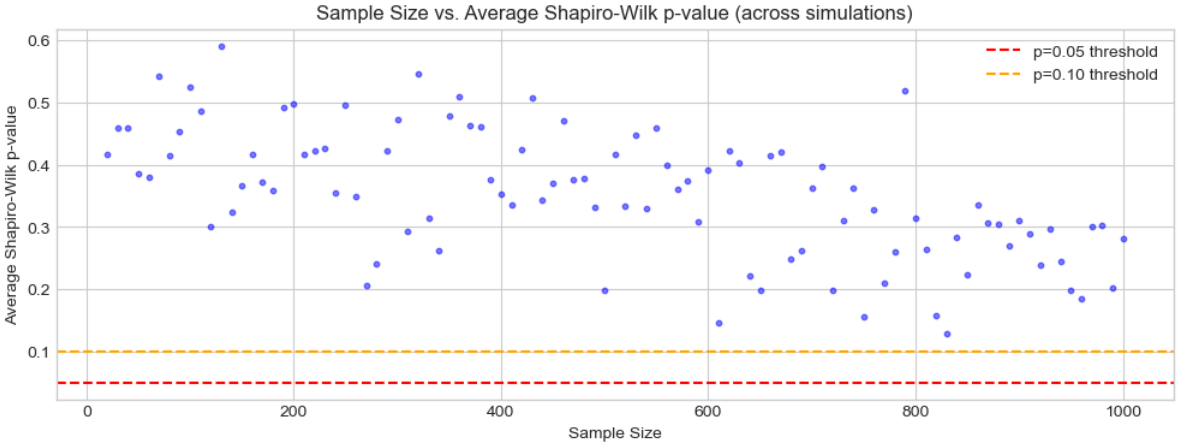
In [124…]
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [125…]
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



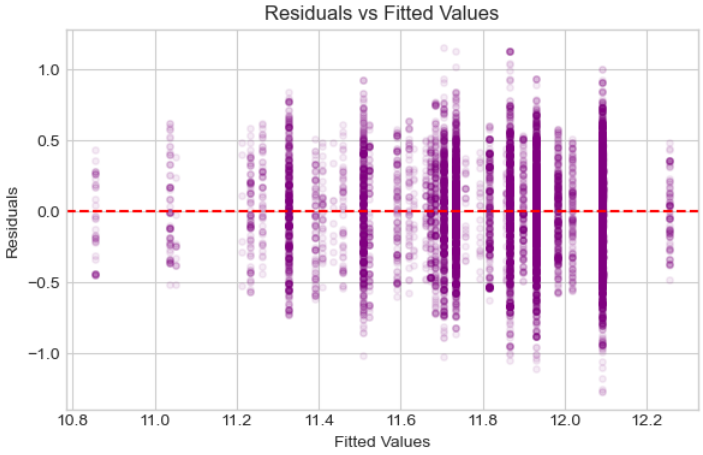


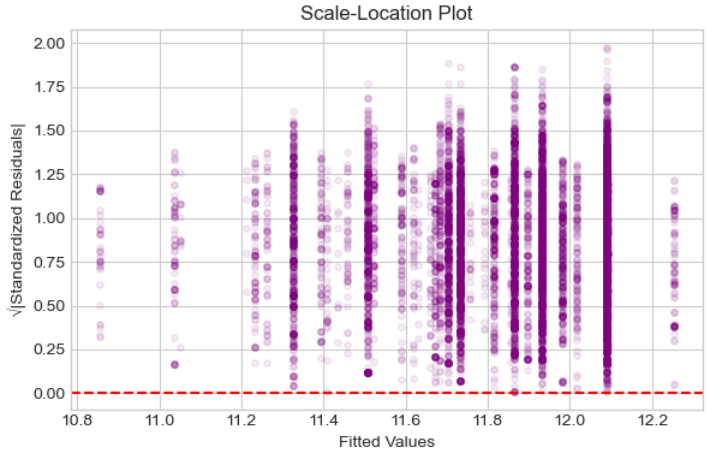
In [126…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



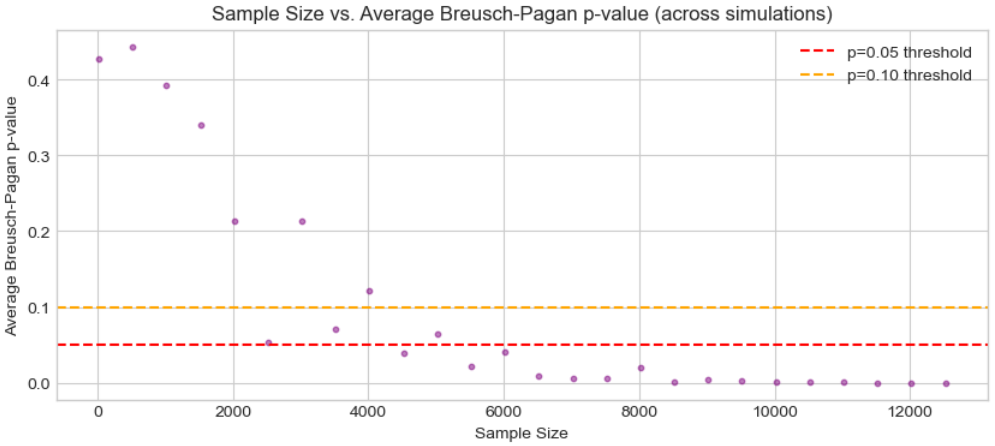
Checking Residuals: Homoscedasticity

In [128…
```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```





In [129…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=13000, step_size=500, num_simulations=10, dot_size=10)
```



Making a prediction

In [131…
```
specific_case = {
    'job_category': 'Data Scientist/ ML Engineer',
    'seniority_level': 'senior',
    'country': 'us'
}
```

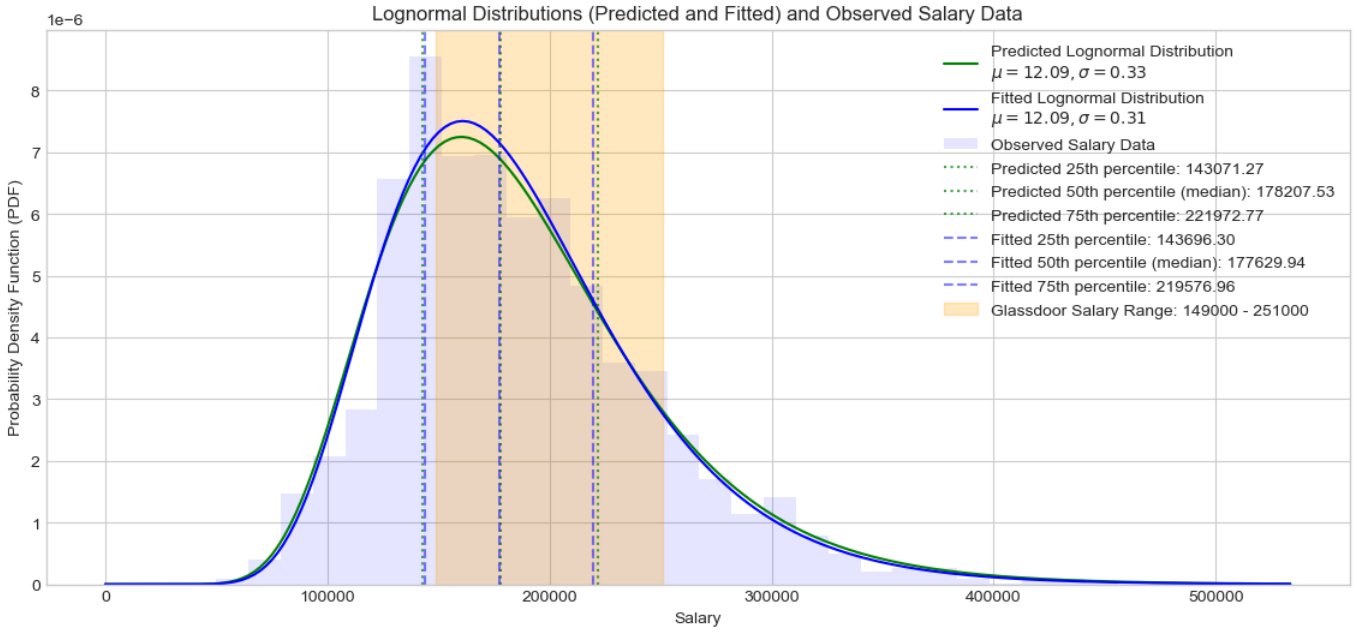
In [132…
```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 178207.53
Predicted 25th Percentile Salary: 143071.27
Predicted 75th Percentile Salary: 221972.77
```

Plotting the prediction

In [134…
```
specific_case = {
    'job_category': 'Data Scientist/ ML Engineer',
    'seniority_level': 'senior',
    'country': 'us'
}
```

In [135…
```
plot_salary_distributions(df_name, model, encoded_features, specific_case, glassdoor_data)
```



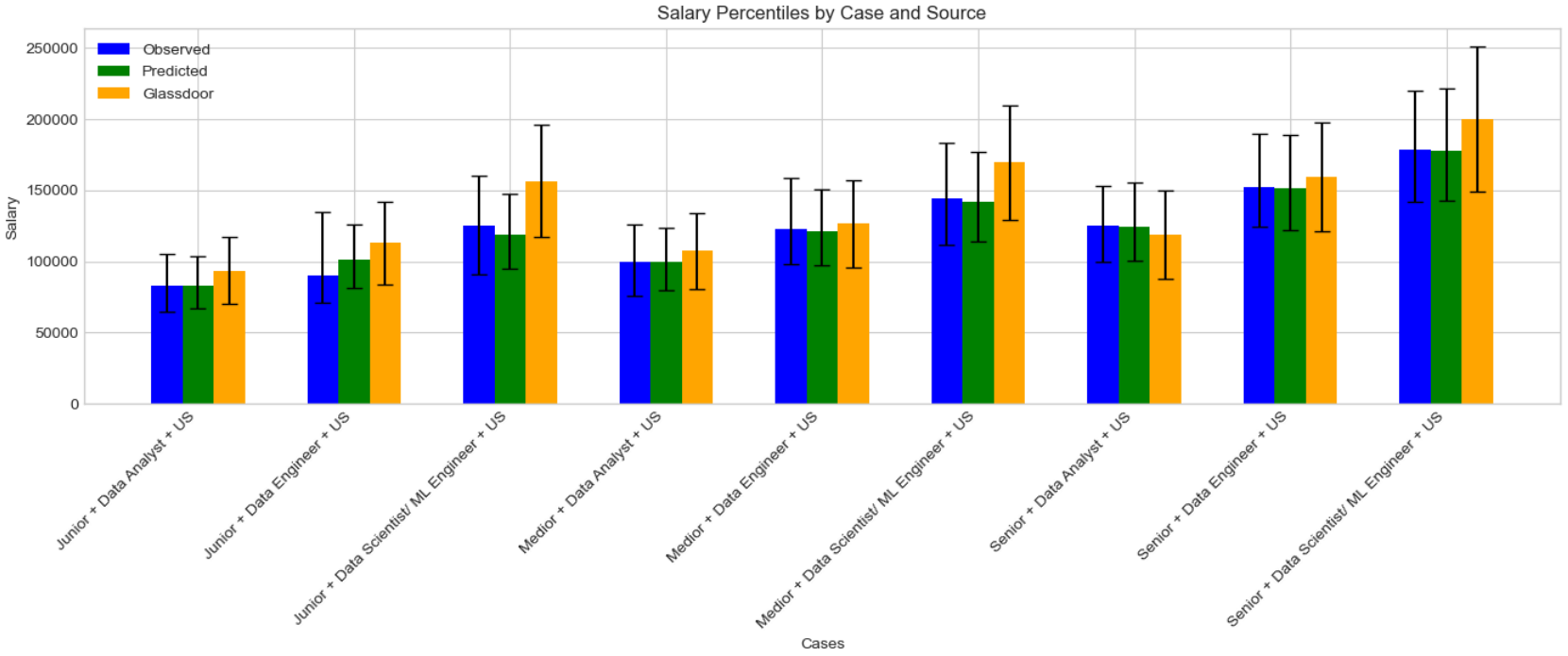
The result

In [137…

```
# Define your specific cases in the desired order
specific_cases = [
    {'job_category': 'Data Analyst', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Analyst', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Analyst', 'seniority_level': 'senior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Engineer', 'seniority_level': 'senior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'junior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'medior', 'country': 'us'},
    {'job_category': 'Data Scientist/ ML Engineer', 'seniority_level': 'senior', 'country': 'us'},
]
```

In [138…

```
plot_salary_percentiles_by_case(df_name, model, encoded_features, specific_cases, glassdoor_data)
```



Extended model

Initial run

In [533…

```
df_name = df_it_w_l
categorical_vars = ['job_category','seniority_level','country','language_category','company_size_category','city_category','industry_category']

specific_case = {
    'job_category': 'Data Analyst',
    'seniority_level': 'senior',
    'country': 'de',
    'language_category': 'English-speaking (but not german)',
    'company_size_category': 'l',
    'city_category': 'munich',
    'industry_category': 'manufacturing, transportation, or supply chain'
}
```

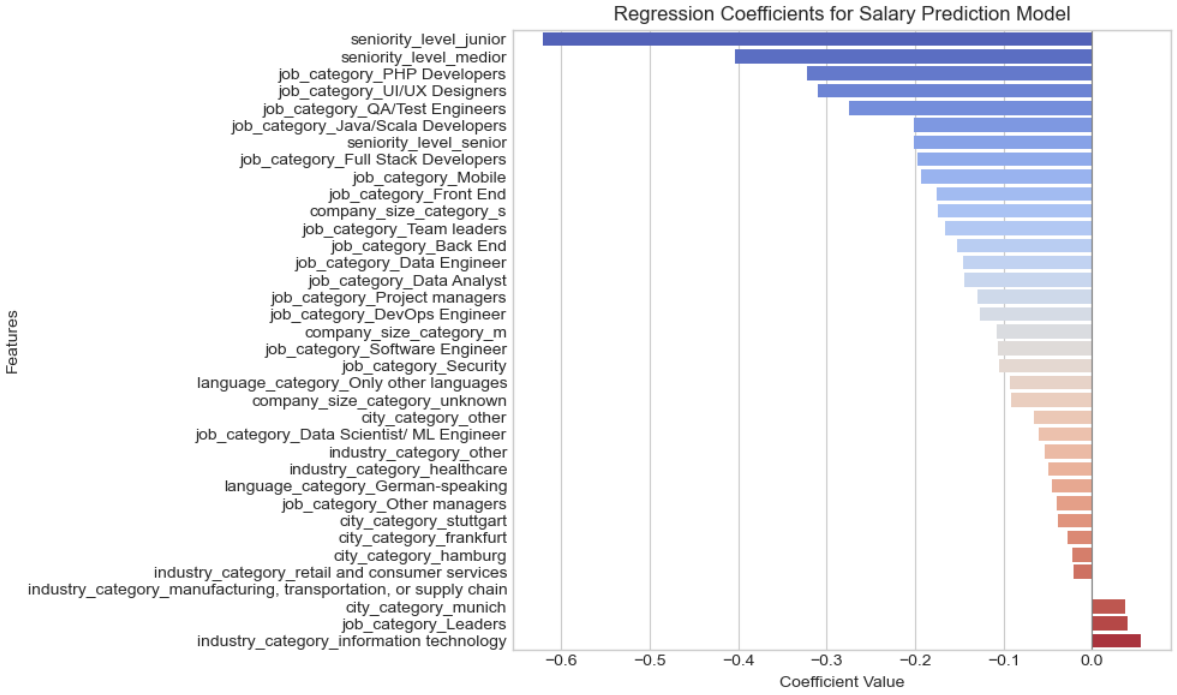
In [535…

```
# Your prepared DataFrame
X = df_name[categorical_vars]
y = df_name['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)
model_for_coefficients_02 = model

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.486
Model:                            OLS   Adj. R-squared:                  0.481
Method:                 Least Squares   F-statistic:                     117.4
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        16:13:36   Log-Likelihood:                 772.72
No. Observations:                4513   AIC:                            -1471.
Df Residuals:                    4476   BIC:                            -1234.
Df Model:                          36
Covariance Type:            nonrobust
================================================================================================================================
                                                                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------------------------
const                                                           11.8897      0.031    377.694      0.000      11.828      11.951
job_category_Back End                                           -0.1527      0.021     -7.333      0.000      -0.194      -0.112
job_category_Data Analyst                                       -0.1449      0.029     -5.034      0.000      -0.201      -0.088
job_category_Data Engineer                                      -0.1452      0.032     -4.510      0.000      -0.208      -0.082
job_category_Data Scientist/ ML Engineer                        -0.0607      0.021     -2.822      0.005      -0.103      -0.019
job_category_DevOps Engineer                                    -0.1272      0.023     -5.475      0.000      -0.173      -0.082
job_category_Front End                                          -0.1753      0.022     -7.846      0.000      -0.219      -0.131
job_category_Full Stack Developers                              -0.1977      0.029     -6.739      0.000      -0.255      -0.140
job_category_Java/Scala Developers                              -0.2018      0.030     -6.670      0.000      -0.261      -0.142
job_category_Leaders                                             0.0406      0.025      1.617      0.106      -0.009       0.090
job_category_Mobile                                             -0.1934      0.024     -7.970      0.000      -0.241      -0.146
job_category_Other managers                                     -0.0396      0.024     -1.652      0.099      -0.087       0.007
job_category_PHP Developers                                     -0.3224      0.050     -6.491      0.000      -0.420      -0.225
job_category_Project managers                                   -0.1290      0.024     -5.422      0.000      -0.176      -0.082
job_category_QA/Test Engineers                                  -0.2748      0.023    -12.005      0.000      -0.320      -0.230
job_category_Security                                           -0.1053      0.049     -2.165      0.030      -0.201      -0.010
job_category_Software Engineer                                  -0.1062      0.020     -5.326      0.000      -0.145      -0.067
job_category_Team leaders                                       -0.1666      0.025     -6.764      0.000      -0.215      -0.118
job_category_UI/UX Designers                                    -0.3103      0.043     -7.250      0.000      -0.394      -0.226
seniority_level_junior                                          -0.6204      0.019    -31.966      0.000      -0.659      -0.582
seniority_level_medior                                          -0.4029      0.012    -34.641      0.000      -0.426      -0.380
seniority_level_senior                                          -0.2007      0.010    -19.150      0.000      -0.221      -0.180
language_category_German-speaking                               -0.0449      0.008     -5.702      0.000      -0.060      -0.029
language_category_Only other languages                          -0.0930      0.024     -3.942      0.000      -0.139      -0.047
company_size_category_m                                         -0.1081      0.007    -15.937      0.000      -0.121      -0.095
company_size_category_s                                         -0.1745      0.010    -17.967      0.000      -0.194      -0.155
company_size_category_unknown                                   -0.0910      0.048     -1.909      0.056      -0.184       0.002
city_category_frankfurt                                         -0.0276      0.019     -1.486      0.137      -0.064       0.009
city_category_hamburg                                           -0.0219      0.018     -1.227      0.220      -0.057       0.013
city_category_munich                                             0.0374      0.008      4.831      0.000       0.022       0.053
city_category_other                                             -0.0653      0.009     -6.924      0.000      -0.084      -0.047
city_category_stuttgart                                         -0.0388      0.023     -1.701      0.089      -0.084       0.006
industry_category_healthcare                                    -0.0493      0.054     -0.912      0.362      -0.155       0.057
industry_category_information technology                         0.0552      0.026      2.086      0.037       0.003       0.107
industry_category_manufacturing, transportation, or supply chain 0.0015      0.046      0.033      0.974      -0.088       0.091
industry_category_other                                         -0.0537      0.024     -2.209      0.027      -0.101      -0.006
industry_category_retail and consumer services                  -0.0213      0.036     -0.590      0.555      -0.092       0.050
==============================================================================
Omnibus:                      143.347   Durbin-Watson:                   1.889
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              187.697
Skew:                           0.356   Prob(JB):                     1.75e-41
Kurtosis:                       3.700   Cond. No.                         46.7
==============================================================================

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

In [435…

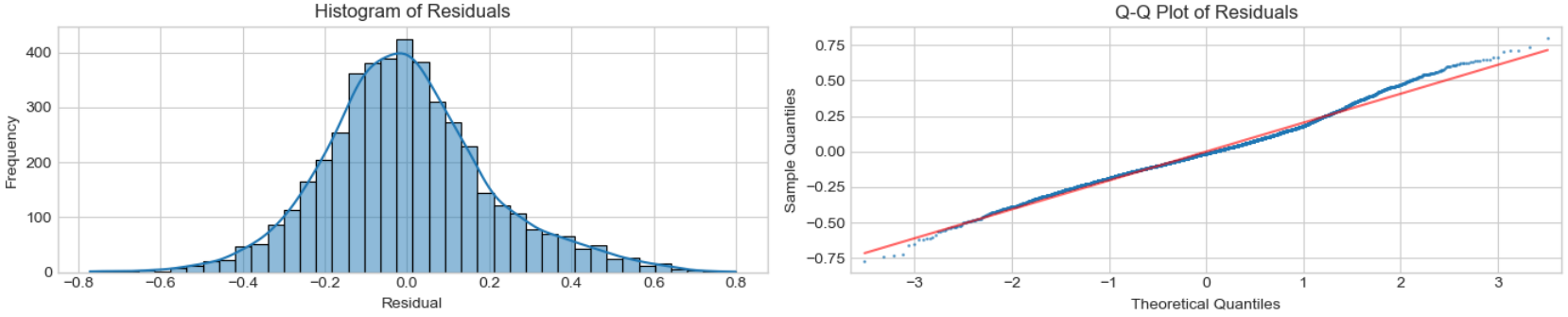
```
plot_model_coefficients(model,10,6)
```



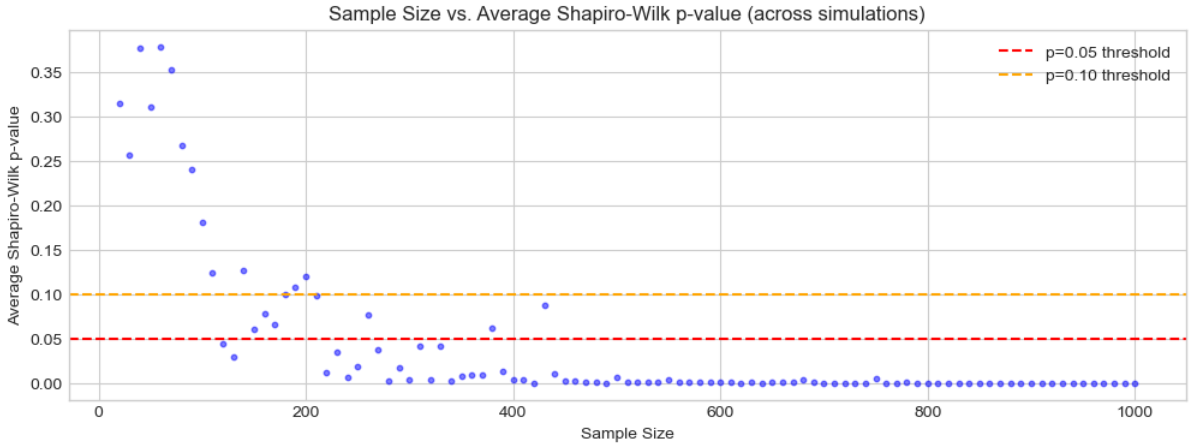
Checking Residuals: Normality

In [146…
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [147…
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



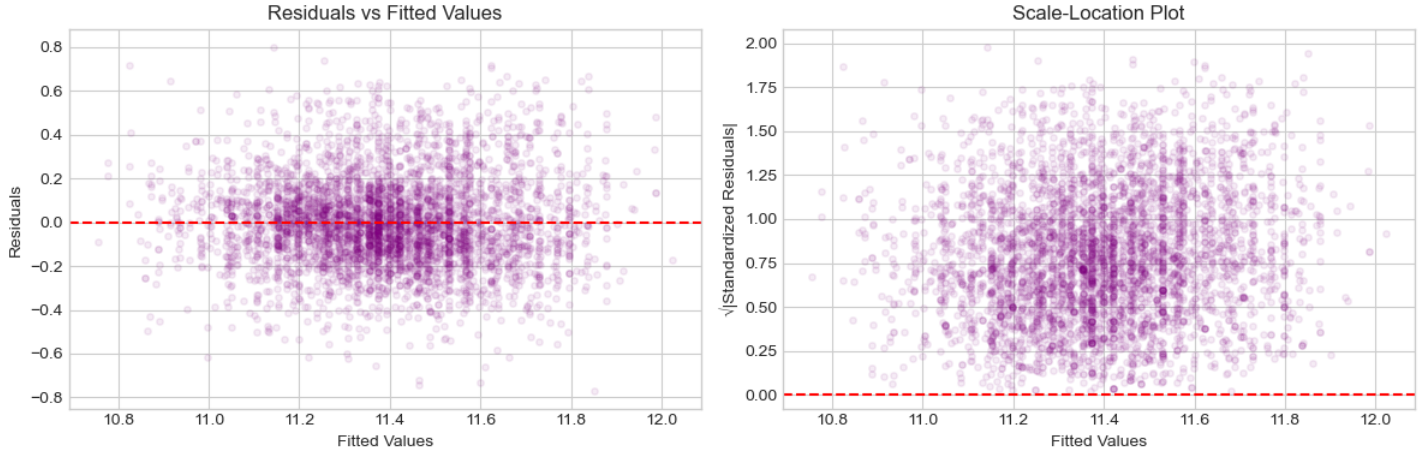
In [148…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



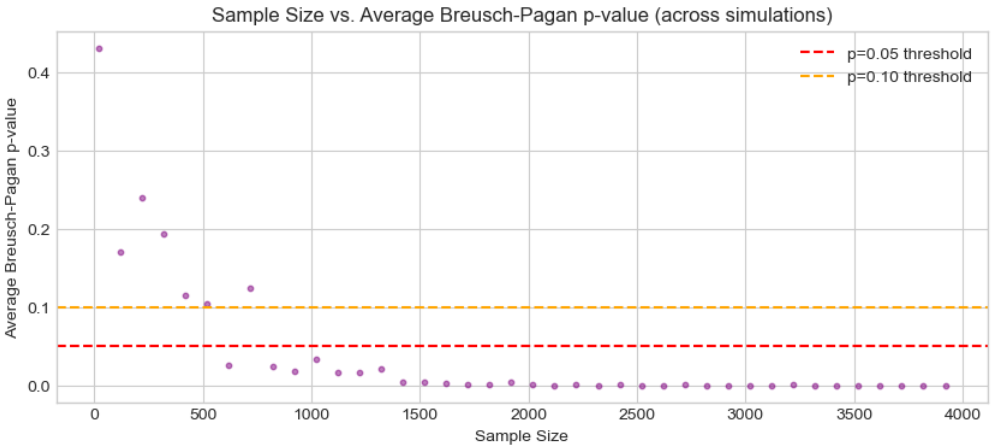
Checking Residuals: Homoscedasticity

In [150…

```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```



In [151…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=4000, step_size=100, num_simulations=10, dot_size=10)
```



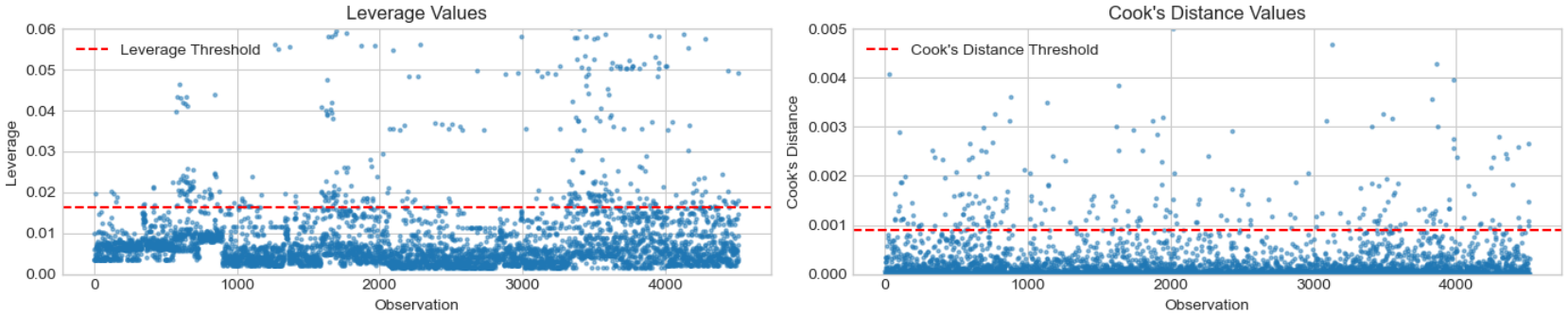
Making an initial prediction

In [153…
```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 107261.12
Predicted 25th Percentile Salary: 93426.52
Predicted 75th Percentile Salary: 123144.35
```

Influential points

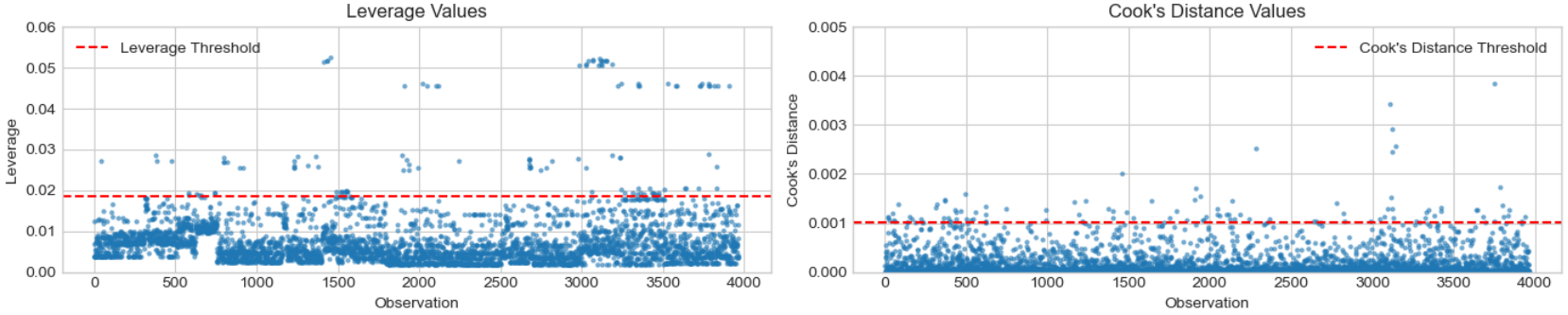
In [155…
```
# Calculate leverage and Cook's Distance using the model
leverage, cooks_d = calculate_leverage_and_cooks_distance(model)

# Plot the leverage and Cook's Distance
plot_leverage_and_cooks_distance(leverage, cooks_d)
```



In [156…
```
# Leverage OR cook's D EITHER high
# Identify observations with high leverage or high Cook's Distance
n = len(leverage)
p = model.df_model
leverage_threshold = 2 * (p + 1) / n
cooks_threshold = 4 / n

# Identify high leverage and high Cook's Distance points separately
high_leverage_points = np.where(leverage > leverage_threshold)[0]
```

```
high_cooks_points = np.where(cooks_d > cooks_threshold)[0]

# Use the union of high leverage and high Cook's Distance points
influential_points = np.union1d(high_leverage_points, high_cooks_points)

print(f"Number of high leverage points: {len(high_leverage_points)}")
print(f"Number of high Cook's Distance points: {len(high_cooks_points)}")
print(f"Number of influential points (either condition): {len(influential_points)}")

# Remove influential observations
X_encoded_cleaned = X_encoded.drop(index=influential_points).reset_index(drop=True)
y_cleaned = y_aligned.drop(index=influential_points).reset_index(drop=True)

# Refit the model without influential observations
model_cleaned = sm.OLS(y_cleaned, X_encoded_cleaned).fit()

# Recalculate Leverage and Cook's Distance
leverage_cleaned, cooks_d_cleaned = calculate_leverage_and_cooks_distance(model_cleaned)

# Plot again
plot_leverage_and_cooks_distance(leverage_cleaned, cooks_d_cleaned)
```

```
Number of high leverage points: 381
Number of high Cook's Distance points: 257
Number of influential points (either condition): 547
```



In [157…

```
# Ensure indices are aligned with the original DataFrame
X_encoded_with_index = X_encoded.reset_index(drop=True)
y_aligned_with_index = y_aligned.reset_index(drop=True)

# Original indices (positions after resetting index)
original_indices = X_encoded_with_index.index

# Map influential points to positions in df_combined
# Reset index of df_combined to ensure alignment
df_name_reset = df_name.reset_index(drop=True)

# Ensure that df_combined_reset has the same number of rows as X_encoded_with_index
assert len(df_name_reset) == len(X_encoded_with_index), "Mismatch in number of rows"

# Retrieve influential observations from the original DataFrame
df_name_influential = df_name_reset.iloc[influential_points]

# Remove the influential points from the original data frame, creating a new df
df_name_noninfl = df_name_reset.drop(index=influential_points).reset_index(drop=True)

# Examine influential observations
#df_name_influential.head()
```

In [158…

```
print("Original DataFrame shape:", df_name_reset.shape)
print("Number of influential points:", len(influential_points))
print("DataFrame without influential points shape:", df_name_noninfl.shape)
```

```
Original DataFrame shape: (4513, 53)
Number of influential points: 547
DataFrame without influential points shape: (3966, 53)
```

comparing VIFs, Running w/o influential points

In [160…

```
# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[160…

	VIF	Feature
0	106.70	Intercept
16	7.98	C(job_category)[T.Software Engineer]
35	6.49	C(industry_category)[T.other]
1	5.73	C(job_category)[T.Back End]
33	5.30	C(industry_category)[T.information technology]
4	4.67	C(job_category)[T.Data Scientist/ ML Engineer]
6	3.58	C(job_category)[T.Front End]
14	3.21	C(job_category)[T.QA/Test Engineers]
21	2.94	C(seniority_level)[T.senior]
5	2.86	C(job_category)[T.DevOps Engineer]
20	2.80	C(seniority_level)[T.medior]
10	2.58	C(job_category)[T.Mobile]
13	2.58	C(job_category)[T.Project managers]
17	2.57	C(job_category)[T.Team leaders]
11	2.53	C(job_category)[T.Other managers]
9	2.35	C(job_category)[T.Leaders]
2	1.80	C(job_category)[T.Data Analyst]
36	1.78	C(industry_category)[T.retail and consumer ser...
7	1.73	C(job_category)[T.Full Stack Developers]
8	1.65	C(job_category)[T.Java/Scala Developers]
3	1.56	C(job_category)[T.Data Engineer]
34	1.39	C(industry_category)[T.manufacturing, transpor...
19	1.39	C(seniority_level)[T.junior]
18	1.26	C(job_category)[T.UI/UX Designers]
32	1.25	C(industry_category)[T.healthcare]
24	1.24	C(company_size_category)[T.m]
25	1.23	C(company_size_category)[T.s]
15	1.18	C(job_category)[T.Security]
12	1.17	C(job_category)[T.PHP Developers]
29	1.15	C(city_category)[T.munich]
30	1.15	C(city_category)[T.other]
22	1.11	C(language_category)[T.German-speaking]
28	1.05	C(city_category)[T.hamburg]
27	1.05	C(city_category)[T.frankfurt]
23	1.04	C(language_category)[T.Only other languages]
31	1.04	C(city_category)[T.stuttgart]
26	1.03	C(company_size_category)[T.unknown]

In [161…

```
# Define the dataframe to be used
df_name = df_name_noninfl

# Define the formula for the model
formula = 'salary_log ~ ' + ' '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')
```

```
# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

Out[161…

	VIF	Feature
0	250.71	Intercept
29	14.84	C(industry_category)[T.other]
28	14.83	C(industry_category)[T.information technology]
14	9.02	C(job_category)[T.Software Engineer]
1	6.56	C(job_category)[T.Back End]
4	5.16	C(job_category)[T.Data Scientist/ ML Engineer]
6	3.90	C(job_category)[T.Front End]
13	3.57	C(job_category)[T.QA/Test Engineers]
5	3.12	C(job_category)[T.DevOps Engineer]
18	2.99	C(seniority_level)[T.senior]
17	2.87	C(seniority_level)[T.medior]
10	2.75	C(job_category)[T.Mobile]
15	2.73	C(job_category)[T.Team leaders]
12	2.73	C(job_category)[T.Project managers]
11	2.69	C(job_category)[T.Other managers]
9	2.42	C(job_category)[T.Leaders]
2	1.81	C(job_category)[T.Data Analyst]
7	1.80	C(job_category)[T.Full Stack Developers]
8	1.62	C(job_category)[T.Java/Scala Developers]
16	1.36	C(seniority_level)[T.junior]
3	1.25	C(job_category)[T.Data Engineer]
22	1.22	C(company_size_category)[T.s]
21	1.22	C(company_size_category)[T.m]
25	1.14	C(city_category)[T.munich]
26	1.13	C(city_category)[T.other]
19	1.09	C(language_category)[T.German-speaking]
24	1.05	C(city_category)[T.hamburg]
23	1.04	C(city_category)[T.frankfurt]
27	1.02	C(city_category)[T.stuttgart]
20	1.02	C(language_category)[T.Only other languages]

In [162…

```
# Your prepared DataFrame
X = df_name_noninfl[categorical_vars]
y = df_name_noninfl['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.527
Model:                            OLS   Adj. R-squared:                  0.524
Method:                 Least Squares   F-statistic:                     151.3
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        14:03:48   Log-Likelihood:                 1158.0
No. Observations:                3966   AIC:                            -2256.
Df Residuals:                    3936   BIC:                            -2067.
Df Model:                          29
Covariance Type:            nonrobust
================================================================================================
                                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                             11.8760      0.046    260.406      0.000      11.787      11.965
job_category_Back End             -0.1503      0.020     -7.361      0.000      -0.190      -0.110
job_category_Data Analyst         -0.1466      0.028     -5.152      0.000      -0.202      -0.091
job_category_Data Engineer        -0.1491      0.043     -3.442      0.001      -0.234      -0.064
job_category_Data Scientist/ ML Engineer -0.0506      0.021     -2.395      0.017      -0.092      -0.009
job_category_DevOps Engineer      -0.1095      0.023     -4.849      0.000      -0.154      -0.065
job_category_Front End            -0.1768      0.022     -8.094      0.000      -0.220      -0.134
job_category_Full Stack Developers -0.2205      0.028     -7.770      0.000      -0.276      -0.165
job_category_Java/Scala Developers -0.2102      0.030     -6.931      0.000      -0.270      -0.151
job_category_Leaders               0.0271      0.025      1.097      0.273      -0.021       0.075
job_category_Mobile               -0.2047      0.024     -8.676      0.000      -0.251      -0.158
job_category_Other managers       -0.0333      0.023     -1.422      0.155      -0.079       0.013
job_category_Project managers     -0.1008      0.023     -4.327      0.000      -0.146      -0.055
job_category_QA/Test Engineers    -0.2802      0.022    -12.599      0.000      -0.324      -0.237
job_category_Software Engineer    -0.1043      0.020     -5.296      0.000      -0.143      -0.066
job_category_Team leaders         -0.1433      0.024     -5.942      0.000      -0.191      -0.096
seniority_level_junior            -0.6123      0.019    -32.283      0.000      -0.650      -0.575
seniority_level_medior            -0.3817      0.011    -34.422      0.000      -0.403      -0.360
seniority_level_senior            -0.1840      0.010    -18.382      0.000      -0.204      -0.164
language_category_German-speaking -0.0418      0.007     -5.579      0.000      -0.057      -0.027
language_category_Only other languages -0.1158      0.046     -2.527      0.012      -0.206      -0.026
company_size_category_m           -0.1097      0.006    -17.222      0.000      -0.122      -0.097
company_size_category_s           -0.1767      0.009    -19.009      0.000      -0.195      -0.159
city_category_frankfurt           -0.0467      0.018     -2.526      0.012      -0.083      -0.010
city_category_hamburg             -0.0401      0.018     -2.262      0.024      -0.075      -0.005
city_category_munich               0.0358      0.007      4.957      0.000       0.022       0.050
city_category_other               -0.0550      0.009     -6.102      0.000      -0.073      -0.037
city_category_stuttgart           -0.0154      0.028     -0.541      0.589      -0.071       0.040
industry_category_information technology  0.0463      0.042      1.099      0.272      -0.036       0.129
industry_category_other           -0.0642      0.041     -1.570      0.116      -0.144       0.016
==============================================================================
Omnibus:                       82.426   Durbin-Watson:                   1.895
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               88.022
Skew:                           0.343   Prob(JB):                     7.70e-20
Kurtosis:                       3.248   Cond. No.                         46.3
==============================================================================

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

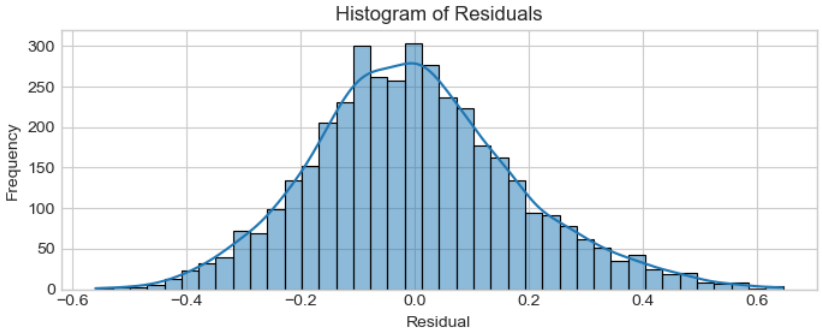
Checking Residuals: Normality

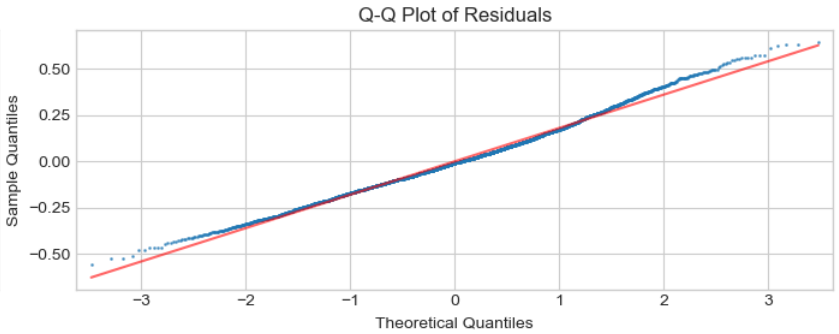
In [164…

```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

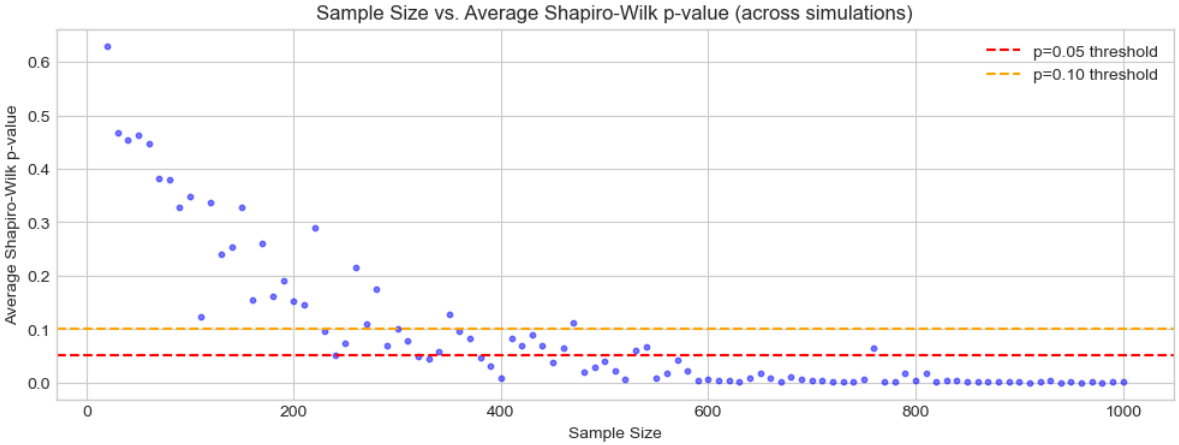
In [165…

```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



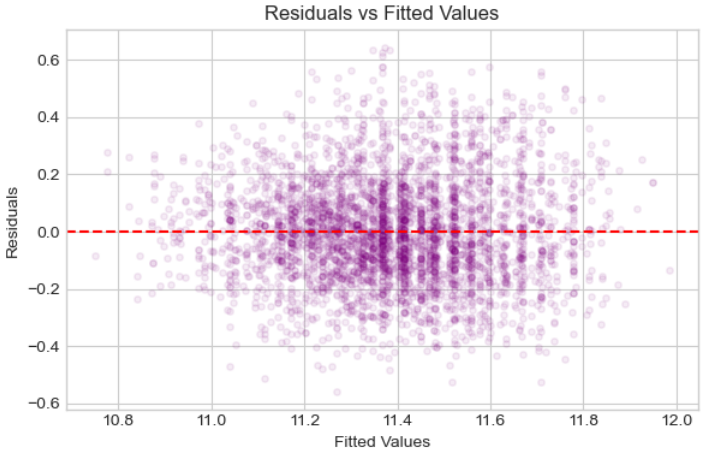


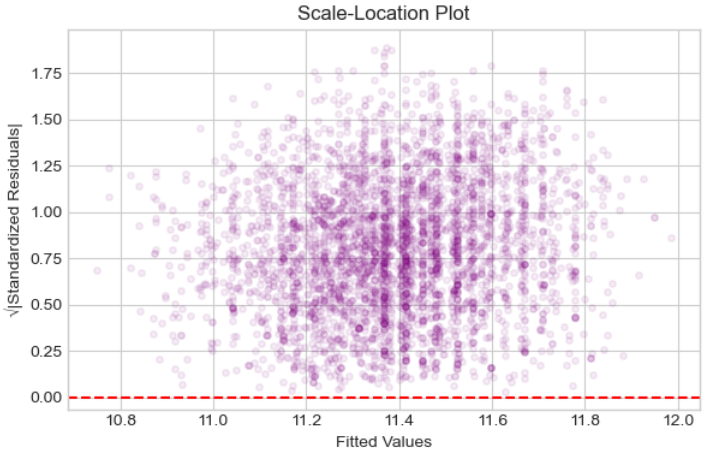
In [166…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



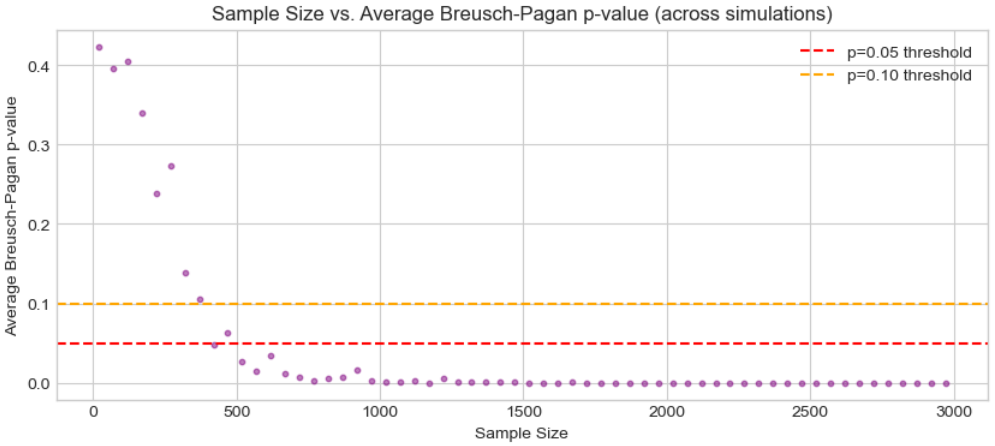
Checking Residuals: Homoscedasticity

In [168…
```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```



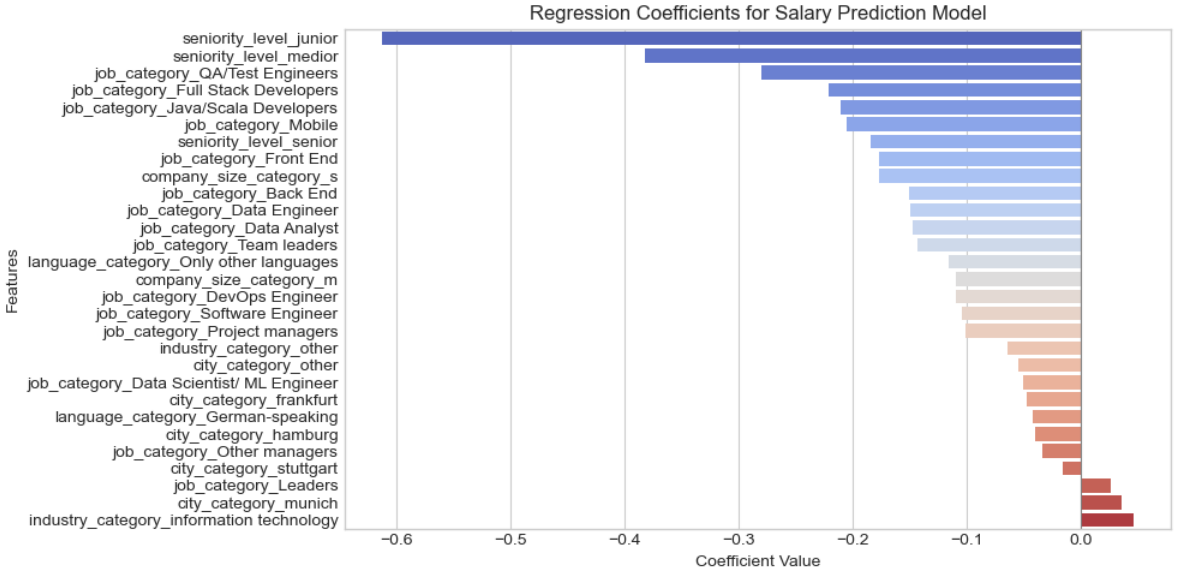


In [169…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=3000, step_size=50, num_simulations=10, dot_size=10)
```



Model interpretation

In [171…
```
plot_model_coefficients(model,10,5)
```



Making an adjusted prediction

In [173…
```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 107072.42
Predicted 25th Percentile Salary: 94742.36
Predicted 75th Percentile Salary: 121007.16
```

Extended model for the combined dataframe

Initial run

In [176…

```
df_name = df_combined
categorical_vars = ['job_category','seniority_level','country','language_category','company_size_category','city_category','industry_category']

specific_case = {
    'job_category': 'Data Analyst',
    'seniority_level': 'senior',
    'country': 'de',
    'language_category': 'English-speaking (but not german)',
    'company_size_category': 'l',
    'city_category': 'munich',
    'industry_category': 'manufacturing, transportation, or supply chain'
}
```

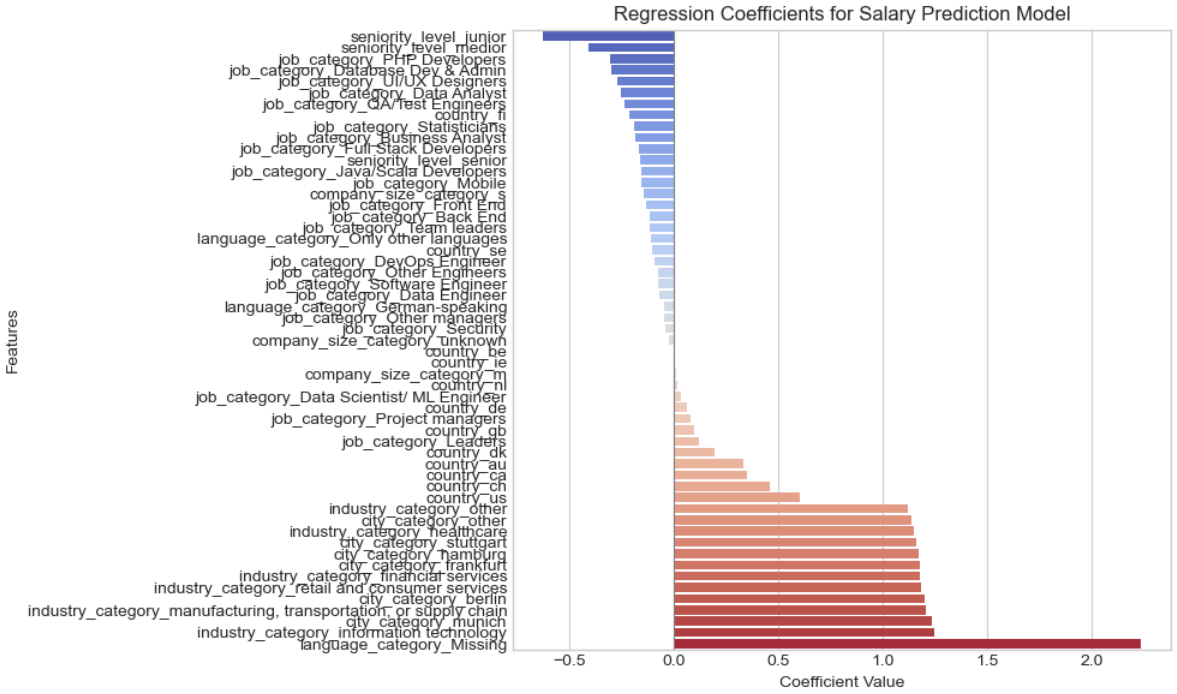
In [488…

```
# Your prepared DataFrame
X = df_name[categorical_vars]
y = df_name['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.486
Model:                            OLS   Adj. R-squared:                  0.481
Method:                 Least Squares   F-statistic:                     117.4
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        16:03:42   Log-Likelihood:                 772.72
No. Observations:                4513   AIC:                            -1471.
Df Residuals:                    4476   BIC:                            -1234.
Df Model:                          36
Covariance Type:            nonrobust
================================================================================================================================
                                                                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------------------------
const                                                           11.8897      0.031    377.694      0.000      11.828      11.951
job_category_Back End                                           -0.1527      0.021     -7.333      0.000      -0.194      -0.112
job_category_Data Analyst                                       -0.1449      0.029     -5.034      0.000      -0.201      -0.088
job_category_Data Engineer                                      -0.1452      0.032     -4.510      0.000      -0.208      -0.082
job_category_Data Scientist/ ML Engineer                        -0.0607      0.021     -2.822      0.005      -0.103      -0.019
job_category_DevOps Engineer                                    -0.1272      0.023     -5.475      0.000      -0.173      -0.082
job_category_Front End                                          -0.1753      0.022     -7.846      0.000      -0.219      -0.131
job_category_Full Stack Developers                              -0.1977      0.029     -6.739      0.000      -0.255      -0.140
job_category_Java/Scala Developers                              -0.2018      0.030     -6.670      0.000      -0.261      -0.142
job_category_Leaders                                             0.0406      0.025      1.617      0.106      -0.009       0.090
job_category_Mobile                                             -0.1934      0.024     -7.970      0.000      -0.241      -0.146
job_category_Other managers                                     -0.0396      0.024     -1.652      0.099      -0.087       0.007
job_category_PHP Developers                                     -0.3224      0.050     -6.491      0.000      -0.420      -0.225
job_category_Project managers                                   -0.1290      0.024     -5.422      0.000      -0.176      -0.082
job_category_QA/Test Engineers                                  -0.2748      0.023    -12.005      0.000      -0.320      -0.230
job_category_Security                                           -0.1053      0.049     -2.165      0.030      -0.201      -0.010
job_category_Software Engineer                                  -0.1062      0.020     -5.326      0.000      -0.145      -0.067
job_category_Team leaders                                       -0.1666      0.025     -6.764      0.000      -0.215      -0.118
job_category_UI/UX Designers                                    -0.3103      0.043     -7.250      0.000      -0.394      -0.226
seniority_level_junior                                          -0.6204      0.019    -31.966      0.000      -0.659      -0.582
seniority_level_medior                                          -0.4029      0.012    -34.641      0.000      -0.426      -0.380
seniority_level_senior                                          -0.2007      0.010    -19.150      0.000      -0.221      -0.180
language_category_German-speaking                               -0.0449      0.008     -5.702      0.000      -0.060      -0.029
language_category_Only other languages                          -0.0930      0.024     -3.942      0.000      -0.139      -0.047
company_size_category_m                                         -0.1081      0.007    -15.937      0.000      -0.121      -0.095
company_size_category_s                                         -0.1745      0.010    -17.967      0.000      -0.194      -0.155
company_size_category_unknown                                   -0.0910      0.048     -1.909      0.056      -0.184       0.002
city_category_frankfurt                                         -0.0276      0.019     -1.486      0.137      -0.064       0.009
city_category_hamburg                                           -0.0219      0.018     -1.227      0.220      -0.057       0.013
city_category_munich                                             0.0374      0.008      4.831      0.000       0.022       0.053
city_category_other                                             -0.0653      0.009     -6.924      0.000      -0.084      -0.047
city_category_stuttgart                                         -0.0388      0.023     -1.701      0.089      -0.084       0.006
industry_category_healthcare                                    -0.0493      0.054     -0.912      0.362      -0.155       0.057
industry_category_information technology                         0.0552      0.026      2.086      0.037       0.003       0.107
industry_category_manufacturing, transportation, or supply chain 0.0015      0.046      0.033      0.974      -0.088       0.091
industry_category_other                                         -0.0537      0.024     -2.209      0.027      -0.101      -0.006
industry_category_retail and consumer services                  -0.0213      0.036     -0.590      0.555      -0.092       0.050
==============================================================================
Omnibus:                      143.347   Durbin-Watson:                   1.889
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              187.697
Skew:                           0.356   Prob(JB):                     1.75e-41
Kurtosis:                       3.700   Cond. No.                         46.7
==============================================================================

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

In [178…

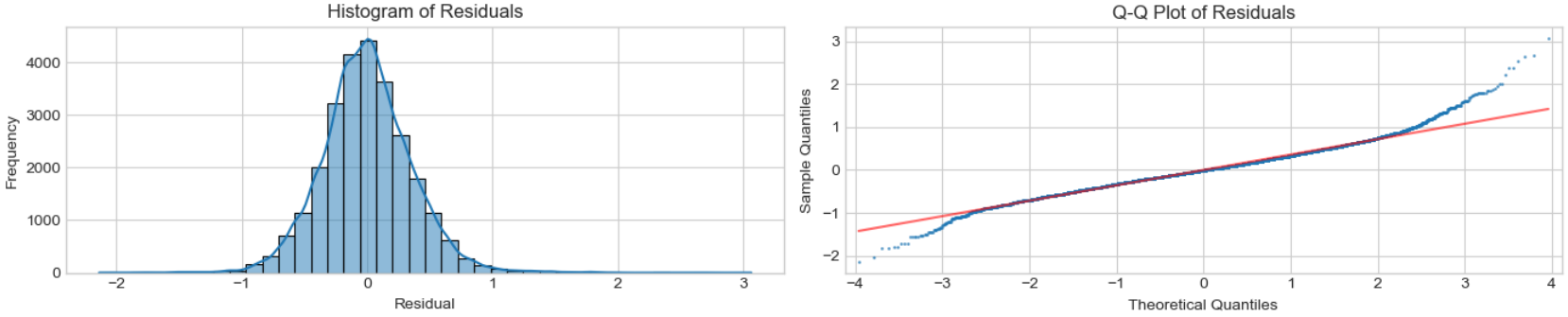
```
plot_model_coefficients(model,10,6)
```



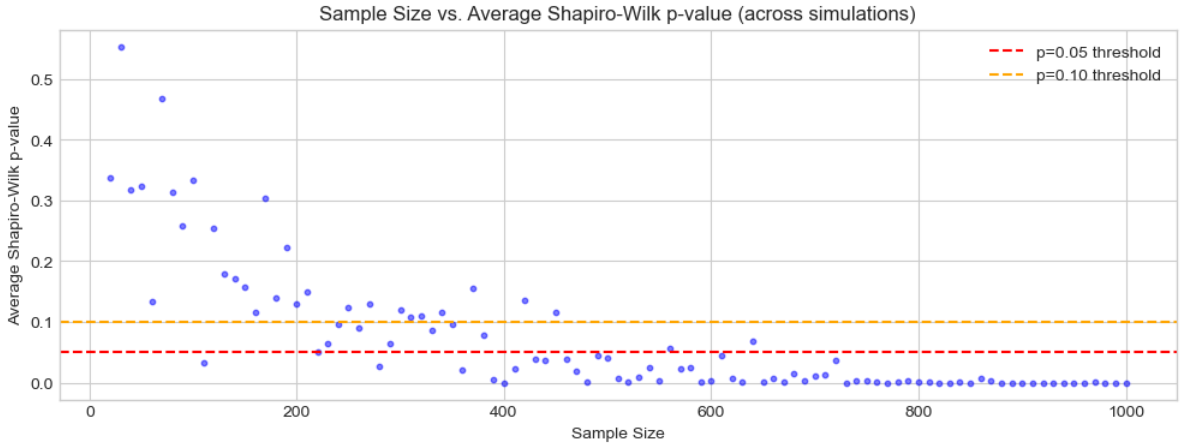
Checking Residuals: Normality

In [180…
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [181…
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



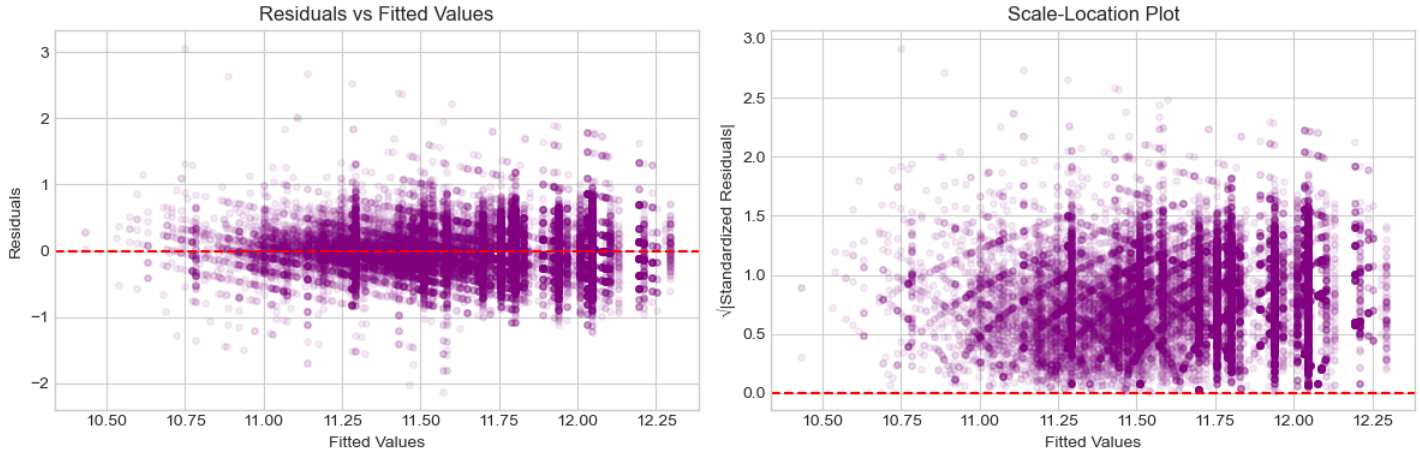
In [182…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



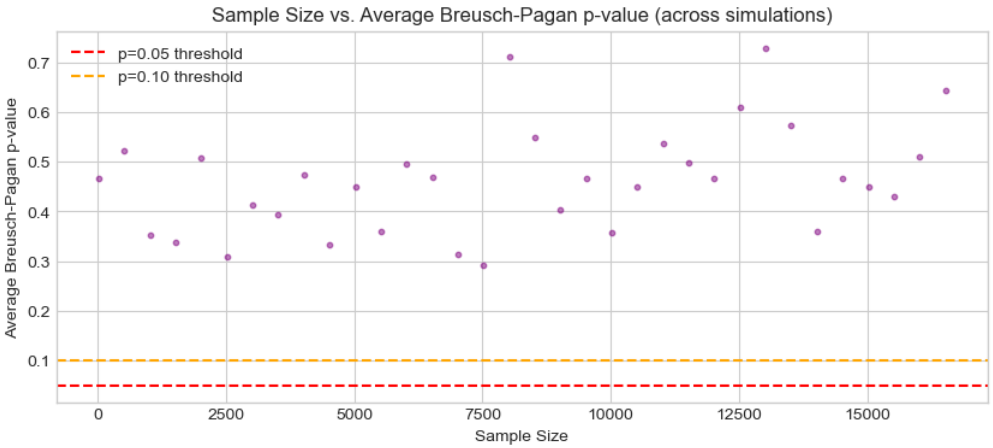
Checking Residuals: Homoscedasticity

In [184…

```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```



In [185…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=17000, step_size=500, num_simulations=10, dot_size=10)
```



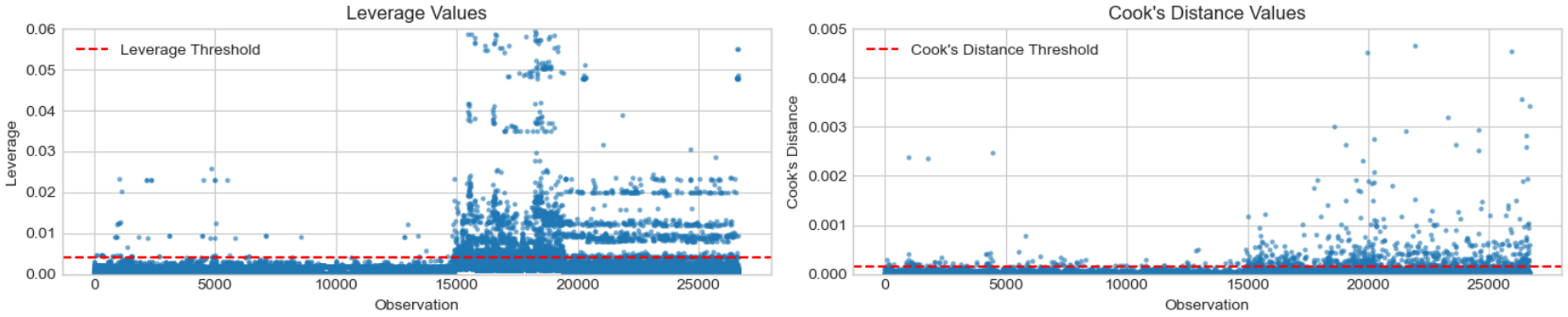
Making an initial prediction

In [187…
```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 89584.85
Predicted 25th Percentile Salary: 70290.72
Predicted 75th Percentile Salary: 114175.04
```

Influential points

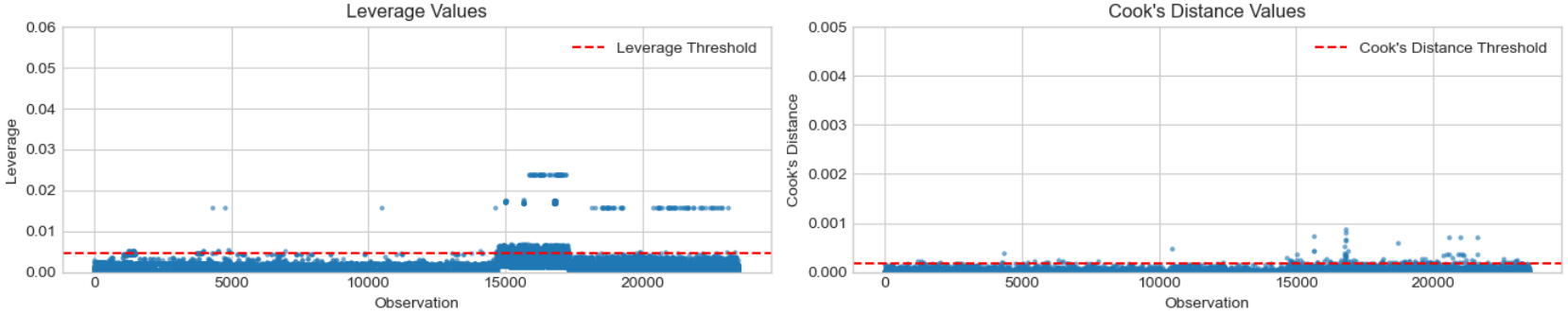
In [189…
```
# Calculate leverage and Cook's Distance using the model
leverage, cooks_d = calculate_leverage_and_cooks_distance(model)

# Plot the leverage and Cook's Distance
plot_leverage_and_cooks_distance(leverage, cooks_d)
```



In [190…
```
# Leverage OR cook's D EITHER high
# Identify observations with high leverage or high Cook's Distance
n = len(leverage)
p = model.df_model
leverage_threshold = 2 * (p + 1) / n
cooks_threshold = 4 / n

# Identify high leverage and high Cook's Distance points separately
high_leverage_points = np.where(leverage > leverage_threshold)[0]
```

```
high_cooks_points = np.where(cooks_d > cooks_threshold)[0]

# Use the union of high leverage and high Cook's Distance points
influential_points = np.union1d(high_leverage_points, high_cooks_points)

print(f"Number of high leverage points: {len(high_leverage_points)}")
print(f"Number of high Cook's Distance points: {len(high_cooks_points)}")
print(f"Number of influential points (either condition): {len(influential_points)}")

# Remove influential observations
X_encoded_cleaned = X_encoded.drop(index=influential_points).reset_index(drop=True)
y_cleaned = y_aligned.drop(index=influential_points).reset_index(drop=True)

# Refit the model without influential observations
model_cleaned = sm.OLS(y_cleaned, X_encoded_cleaned).fit()

# Recalculate Leverage and Cook's Distance
leverage_cleaned, cooks_d_cleaned = calculate_leverage_and_cooks_distance(model_cleaned)

# Plot again
plot_leverage_and_cooks_distance(leverage_cleaned, cooks_d_cleaned)
```

```
Number of high leverage points: 2778
Number of high Cook's Distance points: 884
Number of influential points (either condition): 3151
```



In [191…

```
# Ensure indices are aligned with the original DataFrame
X_encoded_with_index = X_encoded.reset_index(drop=True)
y_aligned_with_index = y_aligned.reset_index(drop=True)

# Original indices (positions after resetting index)
original_indices = X_encoded_with_index.index

# Map influential points to positions in df_combined
# Reset index of df_combined to ensure alignment
df_name_reset = df_name.reset_index(drop=True)

# Ensure that df_combined_reset has the same number of rows as X_encoded_with_index
assert len(df_name_reset) == len(X_encoded_with_index), "Mismatch in number of rows"

# Retrieve influential observations from the original DataFrame
df_name_influential = df_name_reset.iloc[influential_points]

# Remove the influential points from the original data frame, creating a new df
df_name_noninfl = df_name_reset.drop(index=influential_points).reset_index(drop=True)

# Examine influential observations
# df_name_influential.head()
```

In [192…

```
print("Original DataFrame shape:", df_name_reset.shape)
print("Number of influential points:", len(influential_points))
print("DataFrame without influential points shape:", df_name_noninfl.shape)
```

```
Original DataFrame shape: (26655, 63)
Number of influential points: 3151
DataFrame without influential points shape: (23504, 63)
```

comparing VIFs, Running w/o influential points

In [194…

```
# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\statsmodels\regression\linear_model.py:1783: RuntimeWarning: divide by zero encountered in scalar divide
  return 1 - self.ssr/self.centered_tss
C:\ProgramData\anaconda3\Lib\site-packages\statsmodels\regression\linear_model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.centered_tss
```

Out[194...

	VIF	Feature
19	7.98	C(job_category)[T.Software Engineer]
51	6.49	C(industry_category)[T.other]
1	5.73	C(job_category)[T.Back End]
49	5.30	C(industry_category)[T.information technology]
5	4.67	C(job_category)[T.Data Scientist/ ML Engineer]
8	3.58	C(job_category)[T.Front End]
17	3.21	C(job_category)[T.QA/Test Engineers]
25	2.94	C(seniority_level)[T.senior]
7	2.86	C(job_category)[T.DevOps Engineer]
24	2.80	C(seniority_level)[T.medior]
12	2.58	C(job_category)[T.Mobile]
16	2.58	C(job_category)[T.Project managers]
21	2.57	C(job_category)[T.Team leaders]
14	2.53	C(job_category)[T.Other managers]
11	2.35	C(job_category)[T.Leaders]
3	1.80	C(job_category)[T.Data Analyst]
52	1.78	C(industry_category)[T.retail and consumer ser...
9	1.73	C(job_category)[T.Full Stack Developers]
10	1.65	C(job_category)[T.Java/Scala Developers]
4	1.56	C(job_category)[T.Data Engineer]
50	1.39	C(industry_category)[T.manufacturing, transpor...
23	1.39	C(seniority_level)[T.junior]
22	1.26	C(job_category)[T.UI/UX Designers]
48	1.25	C(industry_category)[T.healthcare]
40	1.24	C(company_size_category)[T.m]
41	1.23	C(company_size_category)[T.s]
18	1.18	C(job_category)[T.Security]
15	1.17	C(job_category)[T.PHP Developers]
45	1.15	C(city_category)[T.munich]
46	1.15	C(city_category)[T.other]
38	1.11	C(language_category)[T.German-speaking]
44	1.05	C(city_category)[T.hamburg]
43	1.05	C(city_category)[T.frankfurt]
39	1.04	C(language_category)[T.Only other languages]
47	1.04	C(city_category)[T.stuttgart]
42	1.03	C(company_size_category)[T.unknown]
30	0.00	C(country)[T.de]
0	0.00	Intercept
2	NaN	C(job_category)[T.Business Analyst]
6	NaN	C(job_category)[T.Database Dev & Admin]
13	NaN	C(job_category)[T.Other Engineers]
20	NaN	C(job_category)[T.Statisticians]

	VIF	Feature
26	NaN	C(country)[T.au]
27	NaN	C(country)[T.be]
28	NaN	C(country)[T.ca]
29	NaN	C(country)[T.ch]
31	NaN	C(country)[T.dk]
32	NaN	C(country)[T.fi]
33	NaN	C(country)[T.gb]
34	NaN	C(country)[T.ie]

Note: The warning "invalid value encountered in scalar divide" during VIF calculation likely results from perfect multicollinearity or a lack of variance within certain categorical variables (e.g., constant values, sparse levels, or redundant categories). At this stage, we consider this warning to be informative and indicative of multicollinearity in the selected combination of variables.

In [196…

```
# Define the dataframe to be used
df_name = df_name_noninfl

# Define the formula for the model
formula = 'salary_log ~ ' + ' + '.join(['C(' + var + ')' for var in categorical_vars])

# Get design matrices for VIF calculation
y, X = dmatrices(formula, data=df_name, return_type='dataframe')

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Feature"] = X.columns

vif.sort_values(by='VIF', ascending=False).head(50)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\statsmodels\regression\linear_model.py:1783: RuntimeWarning: divide by zero encountered in scalar divide
  return 1 - self.ssr/self.centered_tss
C:\ProgramData\anaconda3\Lib\site-packages\statsmodels\regression\linear_model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.centered_tss
```

Out[196…

	VIF	Feature
11	8.81	C(job_category)[T.Software Engineer]
1	6.80	C(job_category)[T.Back End]
5	5.48	C(job_category)[T.Data Scientist/ ML Engineer]
6	3.47	C(job_category)[T.Front End]
9	2.98	C(job_category)[T.Project managers]
8	2.87	C(job_category)[T.Other managers]
14	2.65	C(seniority_level)[T.senior]
13	2.58	C(seniority_level)[T.medior]
3	2.11	C(job_category)[T.Data Analyst]
7	1.91	C(job_category)[T.Leaders]
4	1.78	C(job_category)[T.Data Engineer]
10	1.62	C(job_category)[T.QA/Test Engineers]
12	1.40	C(seniority_level)[T.junior]
21	1.21	C(company_size_category)[T.m]
22	1.18	C(company_size_category)[T.s]
24	1.12	C(city_category)[T.other]
23	1.11	C(city_category)[T.munich]
20	1.08	C(language_category)[T.German-speaking]
25	1.05	C(industry_category)[T.other]
16	0.00	C(country)[T.de]
0	0.00	Intercept
2	NaN	C(job_category)[T.Business Analyst]
15	NaN	C(country)[T.ca]
17	NaN	C(country)[T.gb]
18	NaN	C(country)[T.nl]
19	NaN	C(country)[T.us]

Note: The warning "invalid value encountered in scalar divide" during VIF calculation likely results from perfect multicollinearity or a lack of variance within certain categorical variables (e.g., constant values, sparse levels, or redundant categories). At this stage, we consider this warning to be informative and indicative of multicollinearity in the selected combination of variables.

In [198…

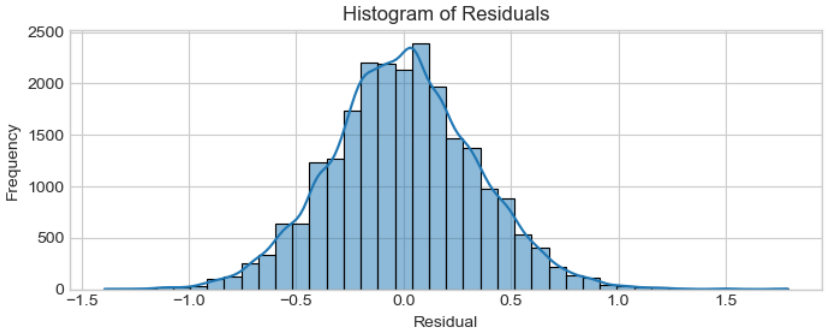
```
# Your prepared DataFrame
X = df_name_noninfl[categorical_vars]
y = df_name_noninfl['salary_log']

# Train the model
model, encoded_features, X_encoded, y_aligned = train_model_1617(X, y)

# View the summary
print(model.summary())
```

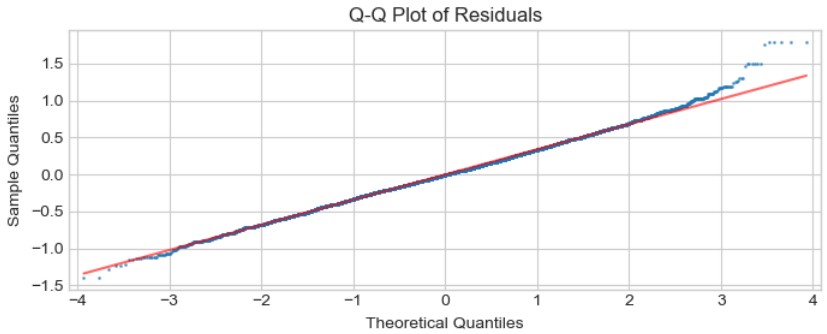
```
                            OLS Regression Results
==============================================================================
Dep. Variable:             salary_log   R-squared:                       0.454
Model:                            OLS   Adj. R-squared:                  0.453
Method:                 Least Squares   F-statistic:                     750.2
Date:                Wed, 11 Dec 2024   Prob (F-statistic):               0.00
Time:                        14:04:01   Log-Likelihood:                -8032.4
No. Observations:               23504   AIC:                         1.612e+04
Df Residuals:                   23477   BIC:                         1.634e+04
Df Model:                          26
Covariance Type:            nonrobust
==========================================================================================================
                                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------
const                                     8.2011      0.020    407.800      0.000       8.162       8.240
job_category_Back End                    -0.1427      0.023     -6.322      0.000      -0.187      -0.098
job_category_Business Analyst            -0.1843      0.021     -8.868      0.000      -0.225      -0.144
job_category_Data Analyst                -0.2600      0.016    -16.391      0.000      -0.291      -0.229
job_category_Data Engineer               -0.0693      0.016     -4.386      0.000      -0.100      -0.038
job_category_Data Scientist/ ML Engineer  0.0456      0.015      2.997      0.003       0.016       0.075
job_category_Front End                   -0.1759      0.030     -5.801      0.000      -0.235      -0.116
job_category_Leaders                      0.1286      0.027      4.772      0.000       0.076       0.181
job_category_Other managers              -0.0426      0.021     -2.008      0.045      -0.084      -0.001
job_category_Project managers             0.0239      0.020      1.207      0.228      -0.015       0.063
job_category_QA/Test Engineers           -0.3831      0.056     -6.888      0.000      -0.492      -0.274
job_category_Software Engineer           -0.1015      0.017     -5.808      0.000      -0.136      -0.067
seniority_level_junior                   -0.5822      0.011    -51.792      0.000      -0.604      -0.560
seniority_level_medior                   -0.3877      0.010    -40.089      0.000      -0.407      -0.369
seniority_level_senior                   -0.1394      0.009    -15.471      0.000      -0.157      -0.122
country_ca                                0.0485      0.022      2.223      0.026       0.006       0.091
country_de                               -0.2524      0.023    -11.024      0.000      -0.297      -0.207
country_gb                               -0.2170      0.021    -10.513      0.000      -0.257      -0.177
country_nl                               -0.3549      0.047     -7.627      0.000      -0.446      -0.264
country_us                                0.3103      0.019     16.607      0.000       0.274       0.347
language_category_German-speaking        -0.0556      0.019     -2.895      0.004      -0.093      -0.018
language_category_Missing                 3.6089      0.013    275.743      0.000       3.583       3.635
company_size_category_m                   0.0302      0.006      5.176      0.000       0.019       0.042
company_size_category_s                  -0.1563      0.010    -15.452      0.000      -0.176      -0.137
city_category_berlin                      1.5446      0.011    142.191      0.000       1.523       1.566
city_category_munich                      1.5728      0.014    110.144      0.000       1.545       1.601
city_category_other                       1.4748      0.017     87.772      0.000       1.442       1.508
industry_category_information technology  2.3848      0.032     74.910      0.000       2.322       2.447
industry_category_other                   2.2073      0.017    133.019      0.000       2.175       2.240
==============================================================================
Omnibus:                      175.094   Durbin-Watson:                   1.961
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              226.141
Skew:                           0.122   Prob(JB):                     7.84e-50
Kurtosis:                       3.414   Cond. No.                     9.94e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 8.96e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Checking Residuals: Normality

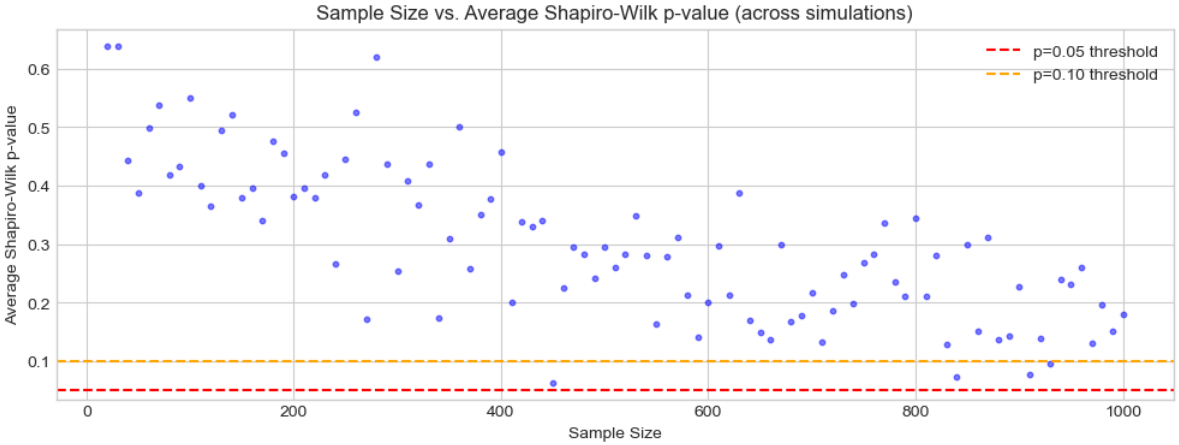
In [200…
```
# Extract residuals from the fitted model
residuals = model.resid
fitted_values = model.fittedvalues
```

In [201…
```
plot_residuals_diagnostics_normality(residuals, hist_bins=40, qq_marker_size=1, qq_alpha=0.6)
```



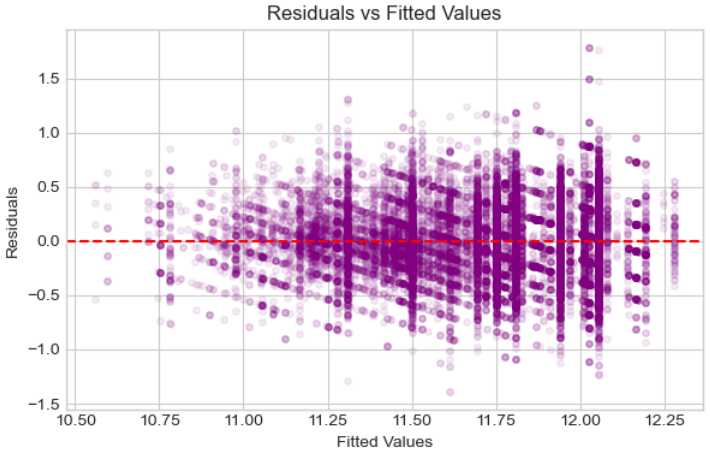


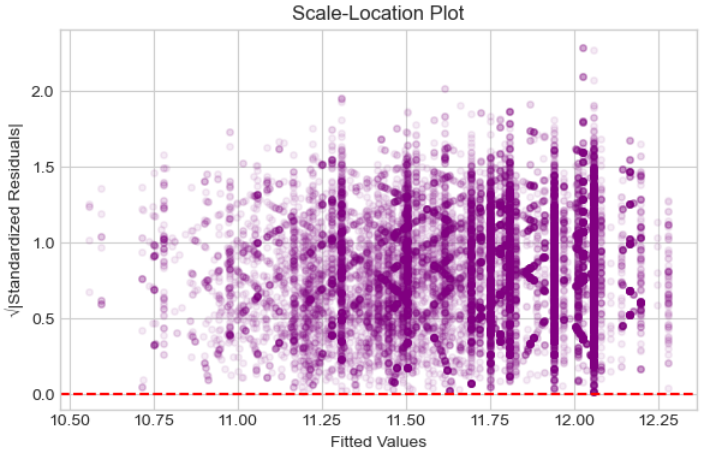
In [202…
```
results_df = simulate_shapiro_test(residuals, min_sample_size=20, max_sample_size=1000, step_size=10, num_simulations=10, dot_size=10)
```



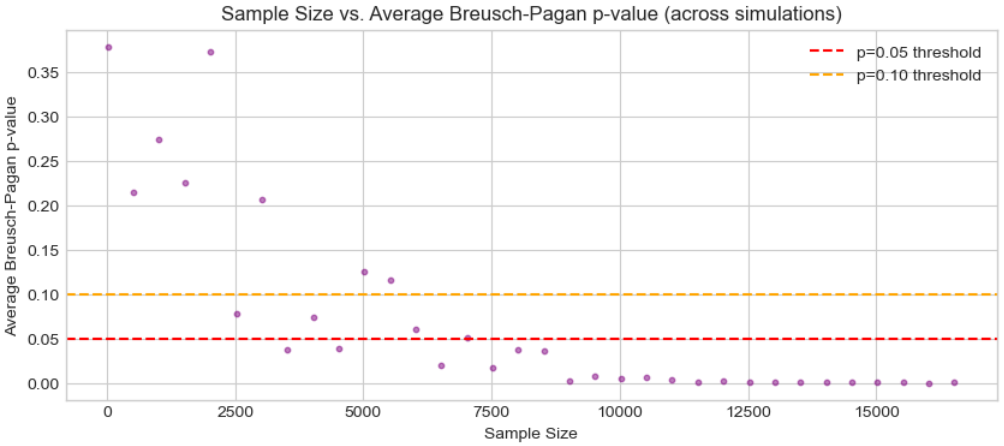
Checking Residuals: Homoscedasticity

In [204…
```
plot_residuals_diagnostics_homoscedasticity(fitted_values, residuals, dot_size=15, alpha=0.08)
```





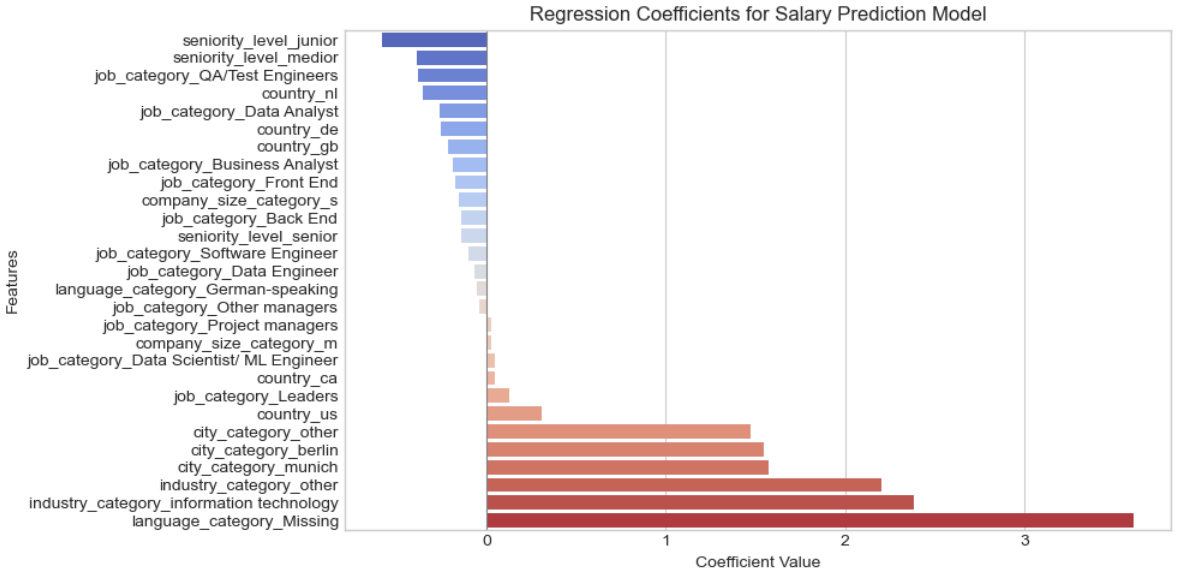
In [205…
```
results_df = simulate_breusch_pagan_test(residuals, fitted_values, min_sample_size=20, max_sample_size=17000, step_size=500, num_simulations=10, dot_size=10)
```



Model interpretation

In [207…

```
plot_model_coefficients(model,10,5)
```



Making an adjusted prediction

In [209…

```
# Predict salary for the specific case
predicted_salary, predicted_salary_p25, predicted_salary_p75 = predict_salary_for_case_log_perc(model, encoded_features, specific_case)

print(f"Predicted Median Salary: {predicted_salary:.2f}")
print(f"Predicted 25th Percentile Salary: {predicted_salary_p25:.2f}")
print(f"Predicted 75th Percentile Salary: {predicted_salary_p75:.2f}")
```

```
Predicted Median Salary: 9154.64
Predicted 25th Percentile Salary: 7274.91
Predicted 75th Percentile Salary: 11520.08
```

The Coefficients

To interpret the model's coefficient, correct VIF values are crucial.

In [557…

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# # Baseline mapping extracted from the data
# baseline_mapping = {}

def transform_coefficients(df, intercept):
    """
    Transform coefficients to percentage differences from the base level (100%).

    Parameters:
    -----------
    df : DataFrame of coefficients
    intercept : float, base intercept value

    Returns:
    --------
    DataFrame with percentage transformations
    """
    # Create a copy of the DataFrame
    transformed_df = df.copy()

    # Convert coefficients to percentages relative to base level (intercept)
    transformed_df['Percentage'] = np.exp(transformed_df['Coefficient'] + np.log(intercept)) / intercept * 100

    return transformed_df

def plot_percentage_coefficients(df, variable_name, intercept, sizex, sizey):
    """
    Plot coefficients as percentage differences from the base level.

    Parameters:
    -----------
    df : DataFrame of coefficients
    variable_name : str, name of the variable to plot
    intercept : float, base intercept value
    """
    # Filter coefficients for the specific variable
    subset = df[df['Variable'] == variable_name].copy()

    # Transform coefficients
    transformed_subset = transform_coefficients(subset, np.exp(intercept))

    # Always include the baseline at 100%, using the baseline_mapping for label
    baseline_label = baseline_mapping.get(variable_name, 'Base/Reference Level')
    baseline_row = pd.DataFrame({
        'Variable': [variable_name],
        'Level': [baseline_label],
        'Percentage': [100.0]
    })

    # Combine baseline with other levels
    plot_df = pd.concat([baseline_row, transformed_subset], ignore_index=True)

    # Sort the levels to make the plot more readable
    plot_df = plot_df.sort_values('Percentage')

    # Create horizontal bar plot
    plt.figure(figsize=(sizex, sizey))
    bars = plt.barh(plot_df['Level'], plot_df['Percentage'], color='#0e2a47ff')

    # Adjust plot limits to ensure labels fit within the background
    max_percentage = plot_df['Percentage'].max()
    plt.xlim(0, max_percentage * 1.2)  # Add padding to the right

    # Add percentage labels to the end of each bar
```

```
    for bar in bars:
        width = bar.get_width()
        plt.text(width, bar.get_y() + bar.get_height()/2, f'{width:.1f}%',
                 ha='left', va='center', fontweight='bold')

    plt.title(f'Percentage Differences for {variable_name} Levels')
    plt.xlabel('Percentage Relative to Base Level')
    plt.ylabel('Levels')
    plt.tight_layout()
    plt.show()
```

From initial model

In [560…

```
# Determine the intercept levels for each categorical variable
baseline_mapping = {}
for var in categorical_vars:
    # Get unique values for the variable, fill NaN with 'Missing', and sort them
    unique_values = sorted(df_combined[var].fillna('Missing').unique())
    baseline_mapping[var] = unique_values[0]  # The first value is the baseline

# Output the baseline mapping
baseline_mapping
```

Out[560…

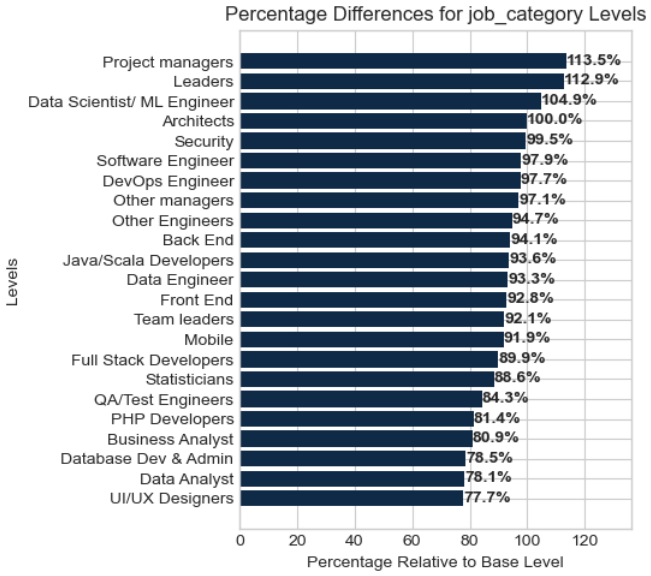
```
{'job_category': 'Architects',
 'seniority_level': 'executive',
 'country': 'at',
 'language_category': 'English-speaking (but not german)',
 'company_size_category': 'l',
 'city_category': 'Missing',
 'industry_category': 'Missing'}
```

In [562…

```
# Example usage (uncomment and adjust as needed)
df, intercept_value = extract_coefficients(model_for_coefficients_01)
```

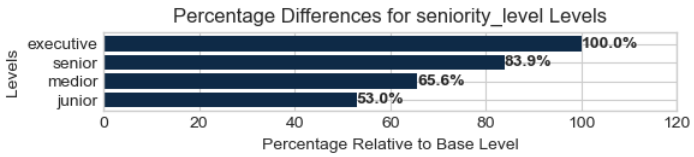
In [564…

```
plot_percentage_coefficients(df, 'job_category', intercept_value, 5.5, 5)
```

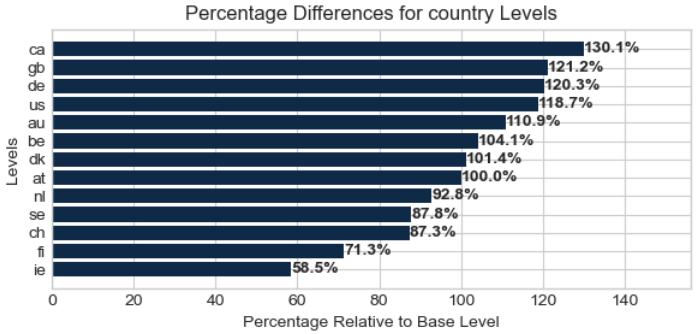


In [572…

```
plot_percentage_coefficients(df, 'seniority_level', intercept_value, 6, 1.5)
```



In [568…
```
plot_percentage_coefficients(df, 'country', intercept_value, 6, 3)
```



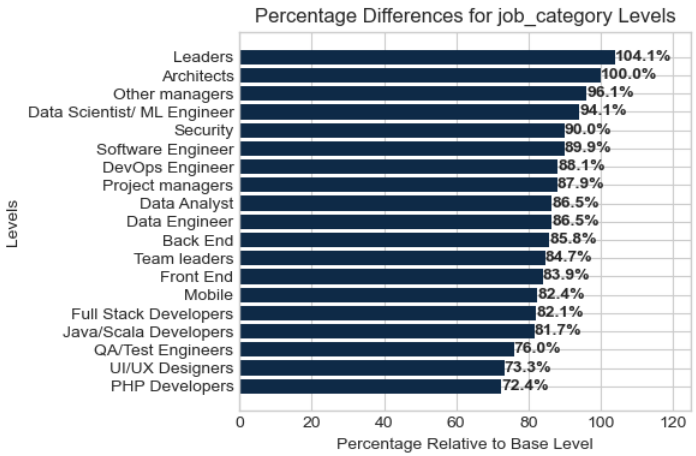
From extended model

In [574…
```
# Determine the intercept levels for each categorical variable
baseline_mapping = {}
for var in categorical_vars:
    # Get unique values for the variable, fill NaN with 'Missing', and sort them
    unique_values = sorted(df_it_w_l[var].fillna('Missing').unique())
    baseline_mapping[var] = unique_values[0]  # The first value is the baseline

# Output the baseline mapping
baseline_mapping
```

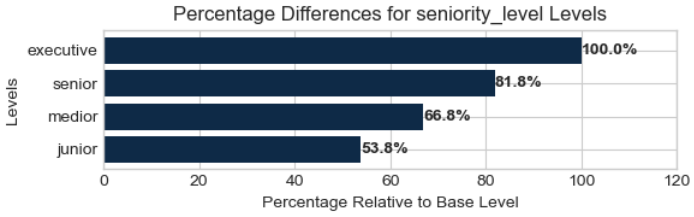
Out[574…
```
{'job_category': 'Architects',
 'seniority_level': 'executive',
 'country': 'de',
 'language_category': 'English-speaking (but not german)',
 'company_size_category': 'l',
 'city_category': 'berlin',
 'industry_category': 'financial services'}
```

In [576…
```
# Example usage (uncomment and adjust as needed)
df, intercept_value = extract_coefficients(model_for_coefficients_02)
```

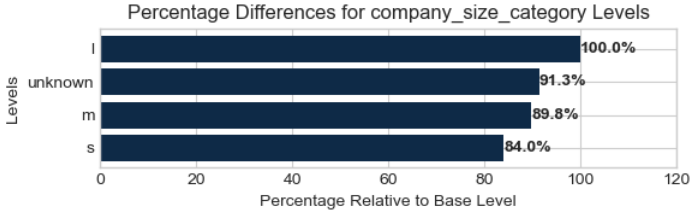
In [578…
```
plot_percentage_coefficients(df, 'job_category', intercept_value, 6, 4)
```



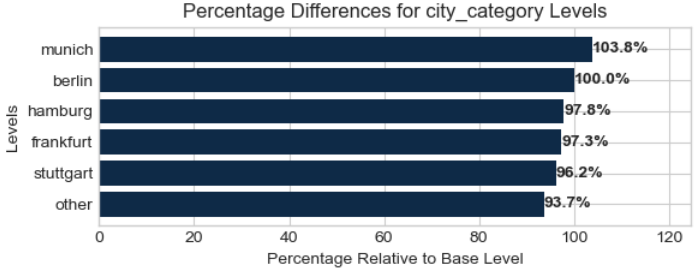
In [580…
```
plot_percentage_coefficients(df, 'seniority_level', intercept_value, 6, 2)
```



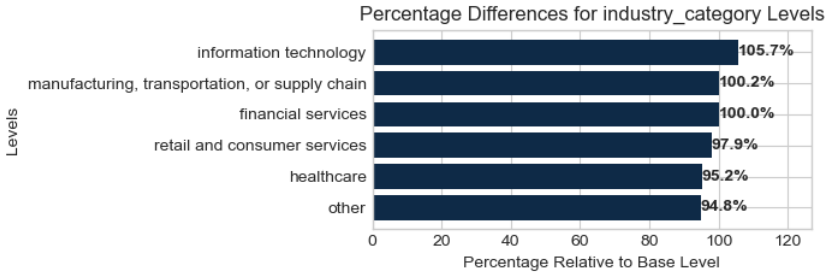
In [582…
```
plot_percentage_coefficients(df, 'company_size_category', intercept_value, 6, 2)
```



In [584…
```
plot_percentage_coefficients(df, 'city_category', intercept_value, 6, 2.5)
```

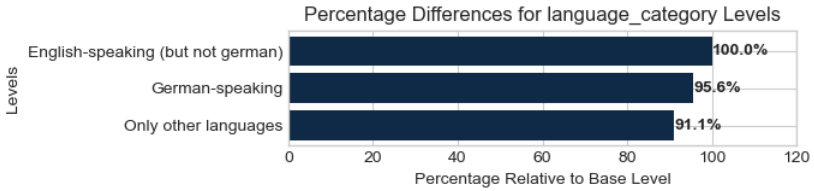


In [586…
```
plot_percentage_coefficients(df, 'industry_category', intercept_value, 7, 2.5)
```



In [595…

```
plot_percentage_coefficients(df, 'language_category', intercept_value, 7, 1.8)
```



Notes

Here are some useful notes on interpreting the concepts/lingo related to predictive models

General interpretation of OLS results

Model Summary
The first section provides an overview of the regression model.
• Dep. Variable: This is the dependent variable (also known as the response or target variable) that the model aims to predict or explain.
It represents the outcome you're studying. For example, if salary_log is the dependent variable, the model predicts the logarithm of salary.
• Model: Indicates the type of regression model used.
OLS stands for Ordinary Least Squares, a method for estimating the unknown parameters in a linear regression model.
• Method: The estimation method used by the model.
Least Squares indicates that the model minimizes the sum of the squares of the residuals (differences between observed and predicted values).
• Date & Time: When the model was fitted.
Useful for record-keeping but doesn't affect model nterpretation.
• No. Observations: The number of observations (sample size) used in the regression.
A larger sample size can provide more reliable estimates.
• Df Residuals: Degrees of freedom of the residuals, calculated as the number of observations minus the number of estimated parameters.
Indicates the amount of information available to estimate the error variance.
• Df Model: Degrees of freedom of the model, equal to the number of independent variables (predictors) in the model.
Shows how many predictors are included, which affects the model's complexity.
• Covariance Type: Specifies the type of covariance matrix used for estimating standard errors.
nonrobust means standard errors assume homoscedasticity (constant variance of errors). Alternative types can adjust for heteroscedasticity.

Model Fit Statistics
These metrics assess how well the model fits the data.
• R-squared: The coefficient of determination, indicating the proportion of variance in the dependent variable explained by the independent variables.
Ranges from 0 to 1. A higher value means a better fit. For example, an R-squared of 0.446 means 44.6% of the variance is explained by the model.
• Adj. R-squared: The adjusted R-squared adjusts the R-squared value based on the number of predictors relative to the sample size.

Accounts for the model complexity. It's useful when comparing models with a different number of predictors.
• F-statistic: Tests the overall significance of the model.
A higher F-statistic suggests that the model explains a significant amount of variance in the dependent variable.
• Prob (F-statistic): The p-value associated with the F-statistic.
A low p-value (typically less than 0.05) indicates that the model is statistically significant.
• Log-Likelihood: A measure of the model's fit based on the likelihood function.
Higher (less negative) values suggest a better fit. Useful for comparing nested models.
• AIC (Akaike Information Criterion): A measure for model comparison that penalizes complexity.
Lower AIC values indicate a better model when comparing different models.
• BIC (Bayesian Information Criterion): Similar to AIC but penalizes model complexity more heavily.
Used for model selection; lower values are preferred.

Coefficients Table
This table provides detailed statistics for each predictor in the model.
• coef (Coefficient): The estimated effect of the predictor on the dependent variable.
Represents the change in the dependent variable for a one-unit change in the predictor, holding other variables constant.
• std err (Standard Error): The standard deviation of the coefficient estimate.
Measures the accuracy of the coefficient estimate. Smaller values indicate more precise estimates.
• t (t-statistic): The test statistic for the hypothesis that the coefficient equals zero.
Calculated as coef / std err. A higher absolute value suggests the coefficient is significantly different from zero.
• P>|t| (P-value): The probability of observing the data if the null hypothesis is true (that the coefficient is zero).
A low p-value (typically less than 0.05) indicates that the predictor is statistically significant.
• [0.025, 0.975] (Confidence Interval): The lower and upper bounds of the 95% confidence interval for the coefficient.
There's a 95% chance the true coefficient lies within this interval.
Example nterpretation for a Predictor Variable:
• Predictor: job_category_Back End
coef: The coefficient indicates the expected change in the dependent variable (e.g., salary_log) when the predictor (job_category_Back End) changes by one unit, holding all other variables constant.
P>|t|: A p-value less than 0.05 suggests the predictor significantly affects the dependent variable.

Diagnostic Tests and Statistics
These metrics help assess the validity of the model assumptions and identify potential issues.
• Omnibus Test: A test for the normality of the residuals.
A significant result (low p-value) suggests the residuals are not normally distributed.
• Prob(Omnibus): The p-value associated with the Omnibus test.
If less than 0.05, it indicates a deviation from normality.
• Jarque-Bera (JB) Test: Another test for normality, focusing on skewness and kurtosis.
Similar to the Omnibus test; a low p-value indicates non-normality.
• Prob(JB): The p-value for the Jarque-Bera test.
• Skew: Measures the asymmetry of the residual distribution.
A value near zero indicates a symmetrical distribution. Positive values indicate right skewness; negative values indicate left skewness.
• Kurtosis: Measures the "tailedness" of the residual distribution.
A value of 3 indicates normal kurtosis. Values greater than 3 suggest heavier tails; values less than 3 suggest lighter tails.
• Durbin-Watson Statistic: Tests for autocorrelation in the residuals.
Values range from 0 to 4. A value around 2 suggests no autocorrelation. Values approaching 0 indicate positive autocorrelation; values approaching 4 indicate negative autocorrelation.
• Cond. No. (Condition Number): Indicates the sensitivity of the regression estimates to small changes in the data, related to multicollinearity.
High values (generally above 30) suggest multicollinearity problems, which can affect the stability of coefficient estimates.

A thought about Normality:
"Necessity of Normality in OLS regression:

Normality of residuals is not strictly required for unbiased coefficients. It is important when: Constructing confidence intervals or p-values (e.g., if n is small, normality matters more). Using methods that depend heavily on normality (e.g., prediction intervals)."

Ensuring Valid Predictions with OLS Regression

1. Verify Model Assumptions
Normality of Residuals: Ensure residuals are approximately normally distributed, as this supports the validity of hypothesis testing and confidence intervals.
Homoscedasticity: Confirm that residuals have constant variance across levels of the predictors. This ensures unbiased and consistent standard errors.
Independence of Observations: Ensure that observations are independent of one another to avoid biases introduced by clustering or temporal dependencies.

2. Handle Outliers and Influential Points
Identify and address leverage points and influential observations using measures such as Cook's Distance or standardized residuals. Removing or adjusting these points can prevent them from disproportionately affecting model results.

3. Address Multicollinearity
Check for multicollinearity using diagnostics such as Variance Inflation Factor (VIF) or condition numbers. High multicollinearity can lead to unstable coefficients and poor interpretability, though predictions may remain valid if the model is properly specified.

4. Ensure Adequate Representation of Categories
For categorical predictors, ensure that each level has a sufficient number of observations. Small sample sizes for certain categories can lead to unstable coefficient estimates and unreliable predictions.

5. Avoid Extrapolation
Limit predictions to combinations of predictors that are well-represented in the training data. Predictions outside this range are less reliable and may reflect model instability rather than true relationships.

6. Evaluate and Include Interaction Effects
Consider whether interactions between predictors are meaningful and test their inclusion. Omitting significant interactions can bias predictions and lead to poor model fit.

7. Validate Model Performance
Evaluate the model's predictive accuracy using techniques such as:
Cross-validation (e.g., k-fold) to assess generalizability. Metrics like R-squared, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) on validation datasets.
Compare training and validation performance to detect overfitting.

8. Address Class Imbalances
For categorical predictors, check for imbalances in levels. Imbalanced data may bias predictions toward dominant categories. Address this through techniques like weighting, oversampling, or stratified validation.

In [ ]: