orig_premium_estimator_main

July 1, 2025

0.0.1 Hyperparameter Tuning

The model will be fine-tuned using `RandomizedSearchCV`, as it is computationally less expensive than `GridSearchCV`.

Hyperparameter combinations are randomly sampled, making this approach generally faster for large search spaces.

RandomizedSearchCV

[114]:
```
xgb = XGBRegressor(booster= 'gbtree')
```

[115]:
```
param_grid = {
    'n_estimators' : [100,200,300],
    'learning_rate' : [0.1,0.15,0.2],
    'max_depth' : [7,8,9],
    'gamma' : [1,2,3]

}
```

A total of 81 hyperparameter combinations are available. A smaller subset of combinations will be selected to reduce computational cost.

[116]:
```
# Adjust based on resources

iter_to_perform = 10
```

[117]:
```
# Initialize RandomizedSearchCV

rscv = RandomizedSearchCV(xgb,param_grid,cv=kf,n_iter=iter_to_perform,random_state=42)
```

[118]:
```
# Executing RandomizedSearchCV and Timing the search

start_time = time()
rscv.fit(features,target)
end_time = time()
total_time_xgb_hyp = end_time - start_time
print(f'Total Time Taken : {round(total_time_xgb_hyp,2)} seconds')
```

```
Total Time Taken : 15.51 seconds
```

Best Model

[119]:
```
# Displaying the results of RandomizedSearchCV execution

pd.DataFrame(rscv.cv_results_)
```

[119]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.197643      0.013730         0.009792        0.001635
1       0.150538      0.007161         0.008529        0.001201
2       0.361103      0.007927         0.016508        0.002543
3       0.349713      0.012096         0.013428        0.001257
4       0.181479      0.052225         0.010587        0.002149
5       0.280001      0.015371         0.014437        0.003418
6       0.280558      0.019303         0.013613        0.002338
7       0.533362      0.050231         0.016778        0.002406
8       0.363804      0.019420         0.014542        0.003370
9       0.207589      0.024729         0.010992        0.001751

   param_n_estimators  param_max_depth  param_learning_rate  param_gamma  \
0                 100                8                 0.10            2
1                 100                7                 0.10            1
2                 200                8                 0.20            1
3                 200                8                 0.10            2
4                 100                7                 0.20            1
5                 200                7                 0.10            2
6                 200                7                 0.15            1
7                 200                9                 0.15            3
8                 200                8                 0.10            1
9                 100                8                 0.15            1

                                              params  split0_test_score  \
0  {'n_estimators': 100, 'max_depth': 8, 'learnin…           0.981313
1  {'n_estimators': 100, 'max_depth': 7, 'learnin…           0.981641
2  {'n_estimators': 200, 'max_depth': 8, 'learnin…           0.979168
3  {'n_estimators': 200, 'max_depth': 8, 'learnin…           0.980615
4  {'n_estimators': 100, 'max_depth': 7, 'learnin…           0.981117
5  {'n_estimators': 200, 'max_depth': 7, 'learnin…           0.981170
6  {'n_estimators': 200, 'max_depth': 7, 'learnin…           0.980671
7  {'n_estimators': 200, 'max_depth': 9, 'learnin…           0.979237
8  {'n_estimators': 200, 'max_depth': 8, 'learnin…           0.980615
9  {'n_estimators': 100, 'max_depth': 8, 'learnin…           0.980905

   split1_test_score  split2_test_score  split3_test_score  split4_test_score  \
0           0.981206           0.981436           0.981245           0.981642
1           0.981450           0.981819           0.981588           0.982032
2           0.978889           0.979135           0.978509           0.979204
3           0.980501           0.980675           0.980507           0.980892
4           0.980785           0.981145           0.981065           0.981333
```

5	0.981046	0.981257	0.981118	0.981439
6	0.980431	0.980752	0.980716	0.980855
7	0.978766	0.979021	0.978330	0.978992
8	0.980501	0.980675	0.980507	0.980892
9	0.980599	0.980815	0.980791	0.981176

	mean_test_score	std_test_score	rank_test_score
0	0.981368	0.000157	2
1	0.981706	0.000201	1
2	0.978981	0.000261	9
3	0.980638	0.000143	7
4	0.981089	0.000177	4
5	0.981206	0.000135	3
6	0.980685	0.000141	6
7	0.978869	0.000308	10
8	0.980638	0.000143	7
9	0.980857	0.000188	5

[120]:
```
# Best score we get from the Tuning

rscv.best_score_
```

[120]: np.float64(0.9817060708999634)

[121]:
```
# Parameters that gave the best score

rscv.best_params_
```

[121]: {'n_estimators': 100, 'max_depth': 7, 'learning_rate': 0.1, 'gamma': 1}

[122]:
```
# Model that resulted the best score

best_model = rscv.best_estimator_
best_model
```

[122]:
```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=1, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.1, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=7,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=100,
             n_jobs=None, num_parallel_tree=None, …)
```

0.0.2 Performance of the 'Best Model'

The best model's performance was evaluated on the test set to assess its real-world effectiveness.

Scores

[123]:
```
# Evaluate Best model's performance
train_score = best_model.score(X_train,y_train)
test_score = best_model.score(X_test,y_test)

# Print the R² scores
print(f'Train Score : {train_score} , Test Score : {test_score} ')
```

```
Train Score : 0.9840264320373535 , Test Score : 0.9839853048324585
```

[124]:
```
# Predict on test data
y_pred = best_model.predict(X_test)

# Calculate Mean Squared Error and Root Mean Squared Error
mse = mean_squared_error(y_test,y_pred)
rmse = root_mean_squared_error(y_test,y_pred)

# Print performance metrics of the best model
print(f'MSE : {mse} , RMSE : {rmse}')
```

```
MSE : 1130643.0 , RMSE : 1063.3170166015625
```

Features and their Importance The contribution of each feature to the model's predictions was analyzed.

[125]:
```
# Retrieve feature names used during model training (available after fitting)

best_model.feature_names_in_
```

[125]:
```
array(['age', 'number_of_dependants', 'income_lakhs', 'insurance_plan',
       'total_risk_score', 'gender_Male', 'region_Northwest',
       'region_Southeast', 'region_Southwest', 'marital_status_Unmarried',
       'bmi_category_Obesity', 'bmi_category_Overweight',
       'bmi_category_Underweight', 'smoking_status_Occasional',
       'smoking_status_Regular', 'employment_status_Salaried',
       'employment_status_Self-Employed'], dtype='<U31')
```

[126]:
```
# Feature importance scores from the fitted model

best_model.feature_importances_
```

[126]:
```
array([2.1001279e-01, 2.0881151e-04, 2.1459715e-04, 7.3175919e-01,
       1.4644399e-02, 1.8761256e-04, 2.0407615e-04, 2.7248557e-04,
       2.3680106e-04, 1.8297366e-04, 1.4747967e-02, 7.1413876e-03,
       5.6649168e-04, 2.0046146e-03, 1.7107116e-02, 2.4060192e-04,
```

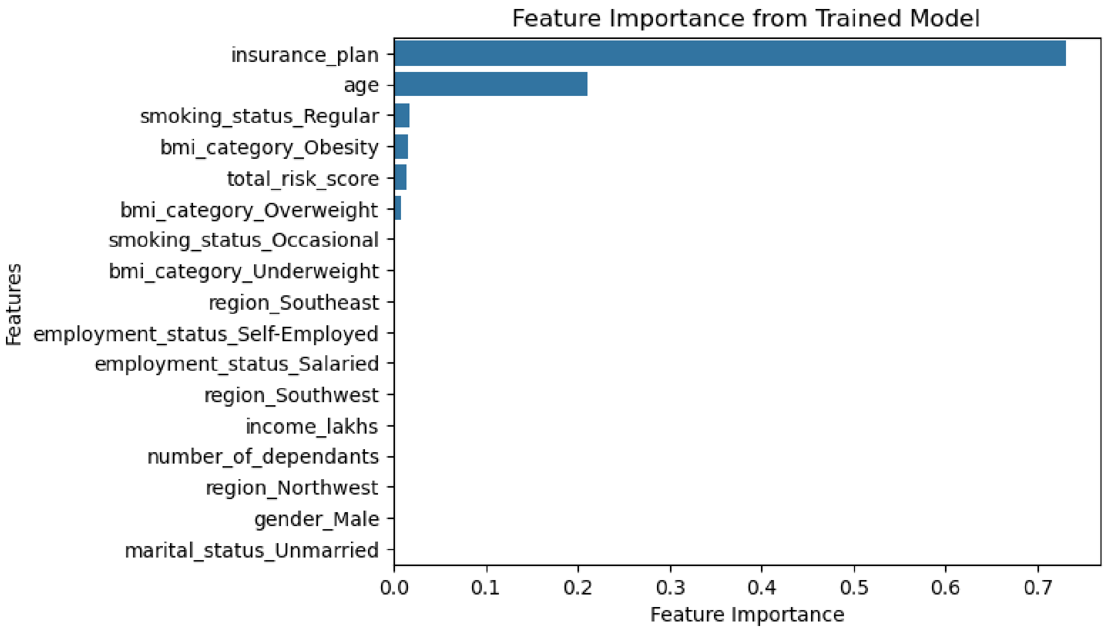
```
 2.6802794e-04], dtype=float32)
```

[127]:
```python
# Create a DataFrame of features and their corresponding importance scores
feat_coef_df = pd.DataFrame(
    {
        'features' : best_model.feature_names_in_,
        'importance' : best_model.feature_importances_
    }
)

# Sort the features by importance in descending order
feat_coef_df = feat_coef_df.sort_values(by=['importance'],ascending=False)
feat_coef_df
```

[127]:

	features	importance
3	insurance_plan	0.731759
0	age	0.210013
14	smoking_status_Regular	0.017107
10	bmi_category_Obesity	0.014748
4	total_risk_score	0.014644
11	bmi_category_Overweight	0.007141
13	smoking_status_Occasional	0.002005
12	bmi_category_Underweight	0.000566
7	region_Southeast	0.000272
16	employment_status_Self-Employed	0.000268
15	employment_status_Salaried	0.000241
8	region_Southwest	0.000237
2	income_lakhs	0.000215
1	number_of_dependants	0.000209
6	region_Northwest	0.000204
5	gender_Male	0.000188
9	marital_status_Unmarried	0.000183

[128]:
```python
# Plot feature importances using a horizontal bar chart
sns.barplot(data = feat_coef_df, x='importance',y='features')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance from Trained Model")
plt.show()
```



0.1 Error Anlysis

The prediction errors (residuals) will be analyzed to evaluate the model's performance.

Objective:
Ensure that 95% of incorrect predictions deviate by 10% from the actual values.

Approach:
- Residuals will be calculated as: residual = actual − predicted
- Percentage error relative to actual values will be computed.
- The proportion of errors falling within the 10% threshold will be checked.
- Any consistent error patterns across customer segments will be identified.

[129]:
```
# Predict on test data

y_pred = best_model.predict(X_test)
```

[130]:
```
# Check the shape of predictions and actual test labels to ensure alignment

print("Predicted labels shape:", y_pred.shape)
print("Actual labels shape:", y_test.shape)
```

```
Predicted labels shape: (14973,)
Actual labels shape: (14973,)
```

0.1.1 Residuals

The difference between the predicted values and the actual values will be determined.

[131]:
```
# Calculate residuals (difference between predicted and actual values)
residuals = y_pred - y_test

# Display the residuals
residuals
```

[131]:
```
24046    -136.548828
199      1481.717773
25415   -2678.477051
32436    -283.119141
30769      17.719727
            …
12098      82.890625
31827     270.870117
6698       71.636719
16918    -222.144531
15081     716.936523
Name: annual_premium_amount, Length: 14973, dtype: float64
```

[132]:
```
# Calculate the residual percentage
# What it shows? - How much the predicted value is deviated from the actual␣
 ↪value
# 2.68 -> Predicted Value is 2.68% higher than the actual value

residuals_pct = ((y_pred - y_test) / y_test)*100
residuals_pct
```

[132]:
```
24046    -1.476842
199      12.703342
25415   -25.577512
32436    -1.077851
30769     0.194594
           …
12098     0.869877
31827     1.902178
6698      0.421591
16918    -0.867075
15081     5.764545
Name: annual_premium_amount, Length: 14973, dtype: float64
```

Everything will be put into a dataframe for better understanding.

[133]:
```
# Create a dictionary to store actual, predicted, residuals, and residual␣
 ↪percentages
residual_dict = {
    'actual' : y_test,
    'predicted' : y_pred,
    'residual' : residuals,
```

```
    'residual_pct' : residuals_pct
}

# Convert the dictionary into a DataFrame for easier analysis
residual_df = pd.DataFrame(residual_dict)

# Display the DataFrame
residual_df
```

[133]:

```
       actual      predicted     residual  residual_pct
24046    9246    9109.451172  -136.548828     -1.476842
199     11664   13145.717773  1481.717773     12.703342
25415   10472    7793.522949 -2678.477051    -25.577512
32436   26267   25983.880859  -283.119141     -1.077851
30769    9106    9123.719727    17.719727      0.194594
…         …            …            …           …
12098    9529    9611.890625    82.890625      0.869877
31827   14240   14510.870117   270.870117      1.902178
6698    16992   17063.636719    71.636719      0.421591
16918   25620   25397.855469  -222.144531     -0.867075
15081   12437   13153.936523   716.936523      5.764545

[14973 rows x 4 columns]
```

[134]:

```
# Sort the residual_df by residual_pct in descending order to see the largest␣
↪errors first

residual_df.sort_values(by=['residual_pct'],ascending=False)
```

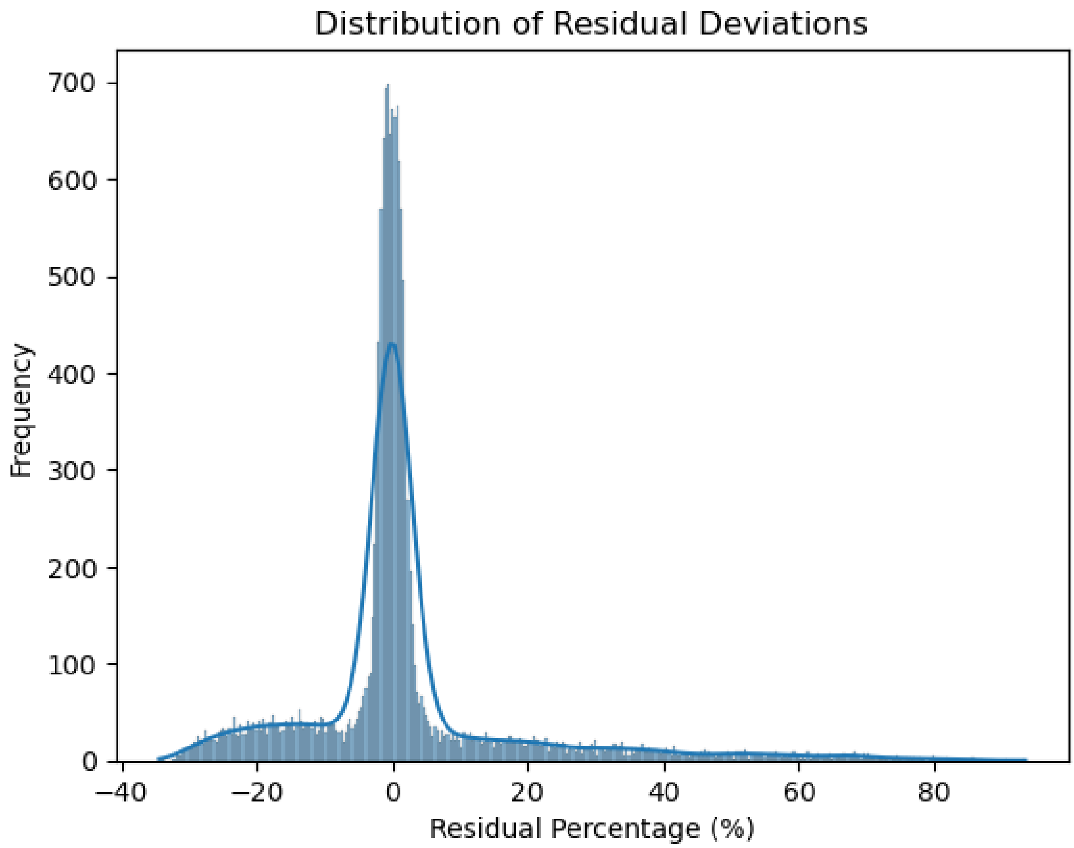
[134]:

```
       actual    predicted     residual  residual_pct
31233    3569  6903.060547  3334.060547     93.417219
23923    3520  6805.419434  3285.419434     93.335779
21960    3627  6908.225586  3281.225586     90.466655
37119    3541  6625.113281  3084.113281     87.097240
13154    3620  6764.991699  3144.991699     86.878224
…         …          …            …           …
42791    9398  6388.824219 -3009.175781    -32.019321
19271    9401  6372.592773 -3028.407227    -32.213671
26266    9420  6384.944336 -3035.055664    -32.219275
25133    9532  6305.118164 -3226.881836    -33.853146
26581    9494  6232.369629 -3261.630371    -34.354649

[14973 rows x 4 columns]
```

[135]:

```
# Plot the distribution of residual percentages with a Kernel Density Estimate␣
↪(KDE) overlay
sns.histplot(data = residual_df, x = 'residual_pct',kde=True )
```

```
plt.title('Distribution of Residual Deviations')
plt.xlabel('Residual Percentage (%)')
plt.ylabel('Frequency')
plt.show()
```



As observed above, some predicted values are 60%, 70%, 80%, or even 100% higher than the actual values, which is undesirable. The instances with such errors will be examined further.

0.1.2 Analysing Extreme Residuals

As stated in the Statement of Work (SOW):

- The goal is to ensure that 95% of incorrect predictions deviate by no more than 10% from the actual values.

This implies that even when predictions are incorrect, **95%** of them should exhibit **less than 10% deviation** - either above or below the actual values.

In other words, only **5%** of incorrect predictions are permitted to deviate by **more than 10%** from the actual values.

Based on this requirement, the deviation threshold has been set at **10%**.

[136]:
```python
# Acceptable percentage deviation allowed between actual and expected values

deviation_pct = 10
```

[137]:
```python
# Filter rows where the absolute residual percentage exceeds the deviation_pct␣
↪(10%)
extreme_residual = residual_df[abs(residual_df['residual_pct']) > deviation_pct]

# Display a random sample of 2 such extreme residuals for inspection
extreme_residual.sample(2)
```

[137]:

	actual	predicted	residual	residual_pct
32744	8911	9975.434570	1064.434570	11.945175
22665	12475	10036.847656	-2438.152344	-19.544307

[138]:
```python
# Get the number of rows and columns in the filtered DataFrame (extreme␣
↪residuals)
extreme_residual.shape
```

[138]:
```
(4204, 4)
```

[139]:
```python
# Get the number of rows and columns in the original residual DataFrame
residual_df.shape
```

[139]:
```
(14973, 4)
```

[140]:
```python
# Calculate the percentage of residuals that exceed the deviation_pct
extreme_residual_percentage = (extreme_residual.shape[0] / residual_df.
↪shape[0])*100

# Display the calculated percentage
extreme_residual_percentage
```

[140]:
```
28.07720563681293
```

It has been observed that approximately **25%** of the predicted values deviate by **more than 10%** from the actual values - **400% more** than the allowed **5%** threshold.

Where this issue originates must be determined.
A deeper investigation is required to understand how such large residuals are produced.

[141]:
```python
# Retrieve the indexes of rows with residuals exceeding the defined threshold

extreme_residual.index
```

[141]:
```
Index([  199, 25415, 47848, 26182, 16869,  5836, 25313,  8385, 38601, 32255,
       …
       15771,  1514, 23737, 24069, 41453, 21846, 10299, 37248, 43739,  7730],
      dtype='int64', length=4204)
```

[142]:
```
len(extreme_residual.index)
```

[142]:
```
4204
```

[143]:
```
# Extract the original rows from X_test where the model had large prediction␣
↪errors
extreme_residual_df = X_test.loc[extreme_residual.index]

# Display the extracted rows with extreme residuals
extreme_residual_df
```

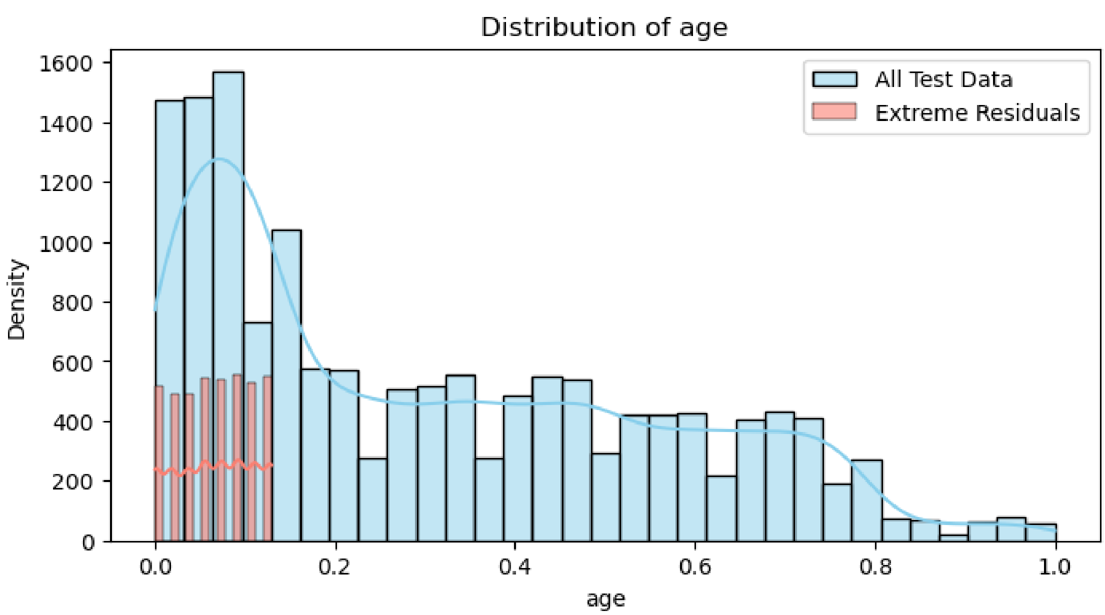
[143]:
```
            age  number_of_dependants  income_lakhs  insurance_plan  \
199    0.129630                   0.2      0.636364             1.0
25415  0.074074                   0.0      0.010101             0.0
47848  0.111111                   0.0      0.424242             1.0
26182  0.037037                   0.0      0.292929             0.0
16869  0.111111                   0.0      0.606061             0.5
…           …                     …           …               …
21846  0.129630                   0.4      0.242424             0.0
10299  0.000000                   0.2      0.181818             0.0
37248  0.018519                   0.4      0.171717             0.0
43739  0.111111                   0.2      0.020202             0.0
7730   0.018519                   0.0      0.313131             0.0

       total_risk_score  gender_Male  region_Northwest  region_Southeast  \
199                 0.0            0                 1                 0
25415               0.0            1                 1                 0
47848               0.0            0                 1                 0
26182               0.0            1                 1                 0
16869               0.0            1                 0                 0
…                     …            …                 …                 …
21846               0.0            1                 0                 1
10299               0.0            1                 0                 1
37248               0.0            0                 0                 1
43739               0.0            0                 0                 1
7730                0.0            0                 1                 0

       region_Southwest  marital_status_Unmarried  bmi_category_Obesity  \
199                   0                         1                     0
25415                 0                         1                     0
47848                 0                         1                     0
26182                 0                         1                     0
16869                 0                         1                     0
…                     …                         …                     …
21846                 0                         1                     1
10299                 0                         1                     1
37248                 0                         0                     0
```

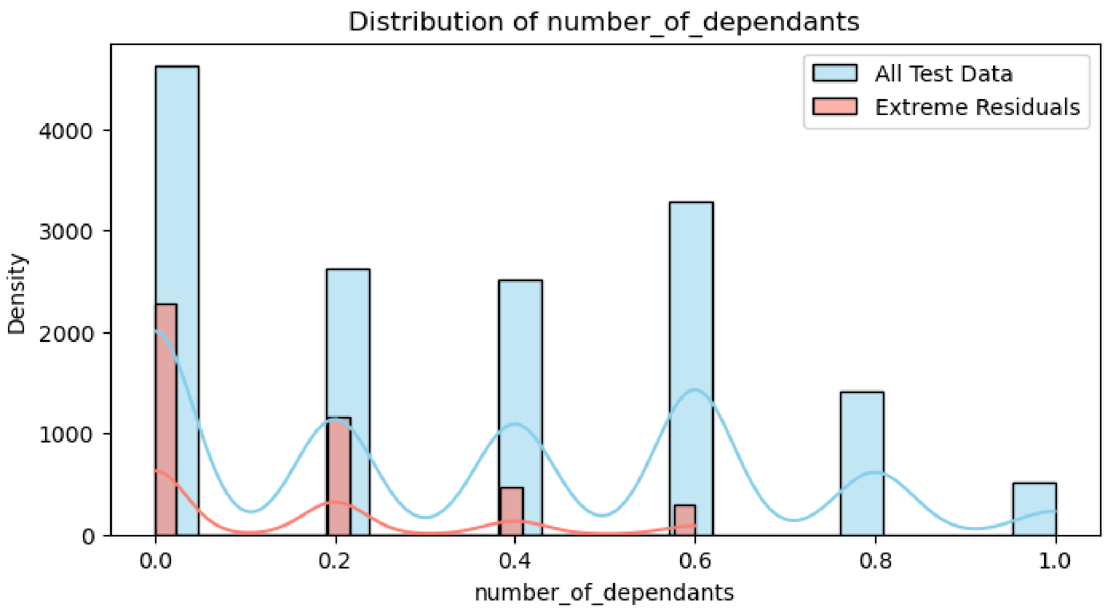
```
43739                    0                         1                      1
7730                     0                         1                      0

       bmi_category_Overweight  bmi_category_Underweight  \
199                          0                         0
25415                        1                         0
47848                        0                         1
26182                        0                         1
16869                        0                         1
…                         …                       …
21846                        0                         0
10299                        0                         0
37248                        0                         0
43739                        0                         0
7730                         0                         0

       smoking_status_Occasional  smoking_status_Regular  \
199                            0                       0
25415                          0                       1
47848                          0                       0
26182                          0                       0
16869                          0                       1
…                           …                     …
21846                          0                       1
10299                          0                       1
37248                          0                       0
43739                          0                       0
7730                           0                       0

       employment_status_Salaried  employment_status_Self-Employed
199                             0                                0
25415                           1                                0
47848                           1                                0
26182                           0                                0
16869                           0                                0
…                            …                              …
21846                           0                                0
10299                           0                                1
37248                           1                                0
43739                           0                                0
7730                            1                                0

[4204 rows x 17 columns]
```

To gain insights about the potential features that causes the large deviations, the distribution of each feature in `X_test` and `extreme_residual_df` will be plotted.

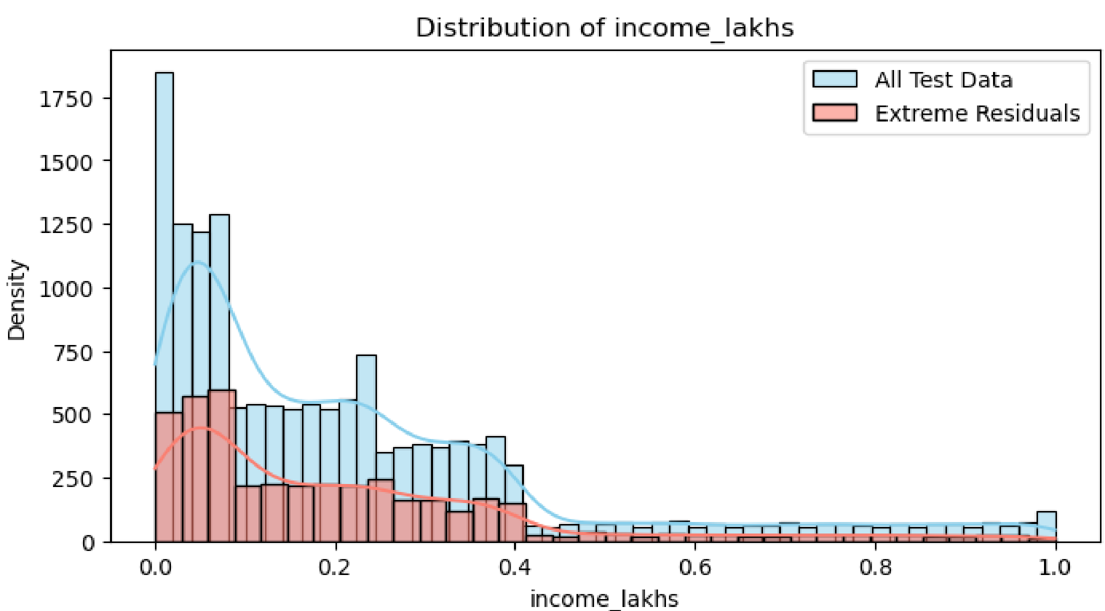
By comparing the distributions across both datasets, it can be observed how each feature behaves and whether any specific feature contributes significantly to the high residuals.

[144]:
```
# Plot feature distributions to compare the X_test set vs rows with extreme␣
↪residuals
for feature in X_test.columns:
    plt.figure(figsize=(8, 4))

    # X_test data distribution
    sns.histplot(data=X_test, x=feature, kde=True, color='skyblue', label='All␣
↪Test Data', alpha=0.5)

    # Extreme residuals distribution
    sns.histplot(data=extreme_residual_df, x=feature, kde=True, color='salmon',␣
↪label='Extreme Residuals', alpha=0.6)

    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.legend()
    plt.show()
```

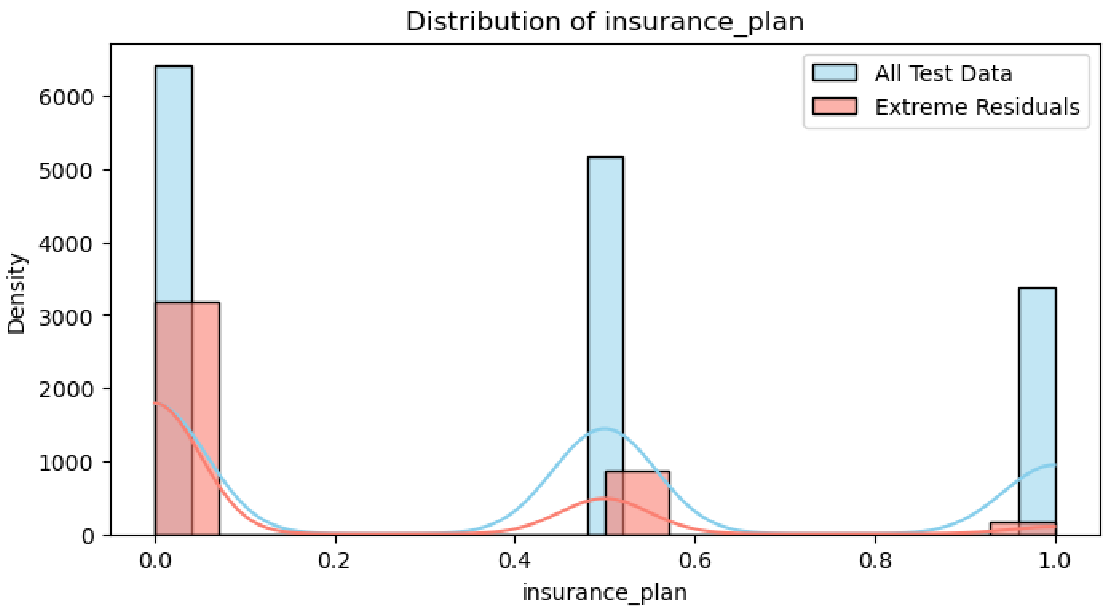




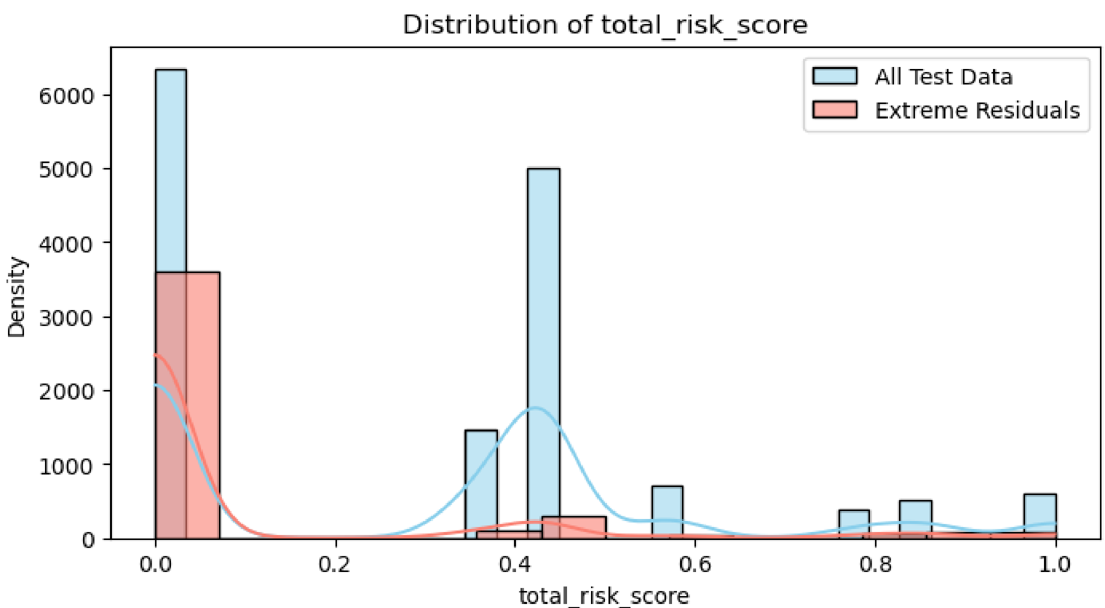
Distribution of number_of_dependants
All Test Data
Extreme Residuals
Density
0
1000
2000
3000
4000
0.0
0.2
0.4
0.6
0.8
1.0
number_of_dependants



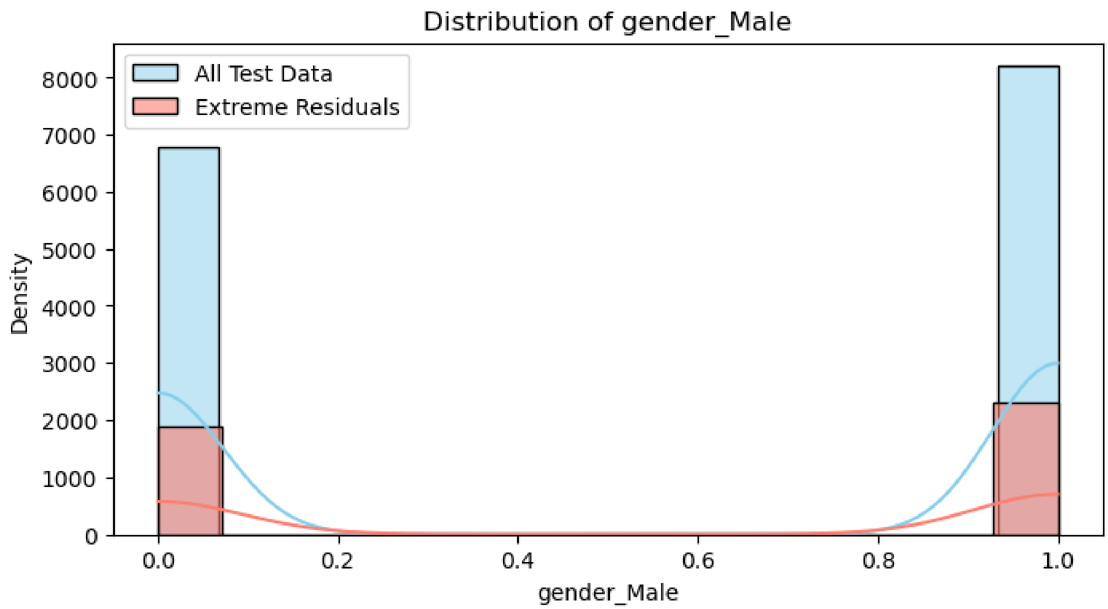
Distribution of income_lakhs
All Test Data
Extreme Residuals
Density
0
250
500
750
1000
1250
1500
1750
0.0
0.2
0.4
0.6
0.8
1.0
income_lakhs

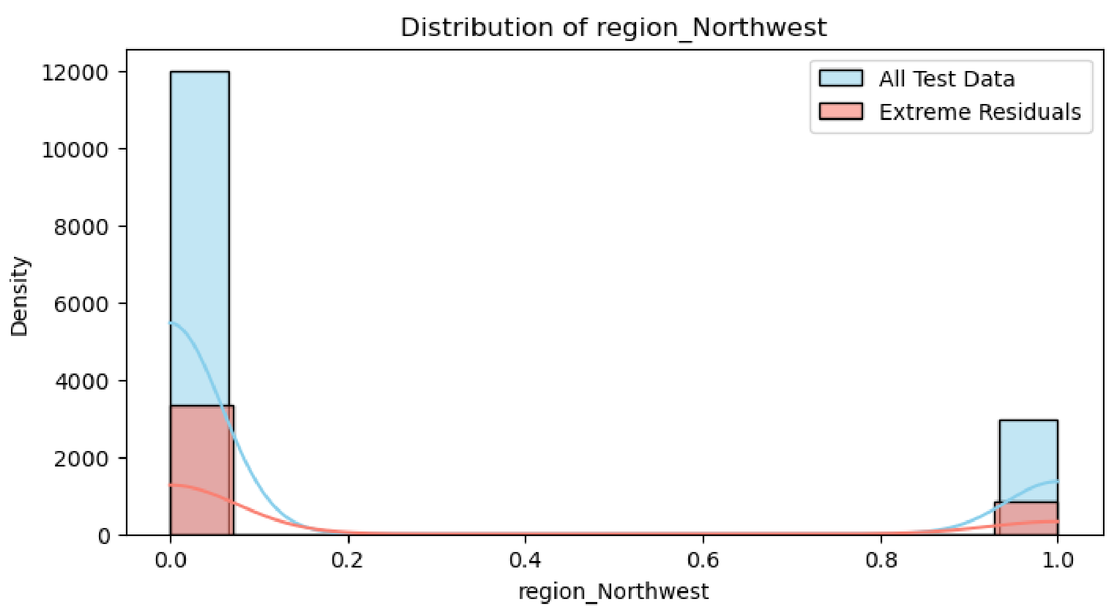


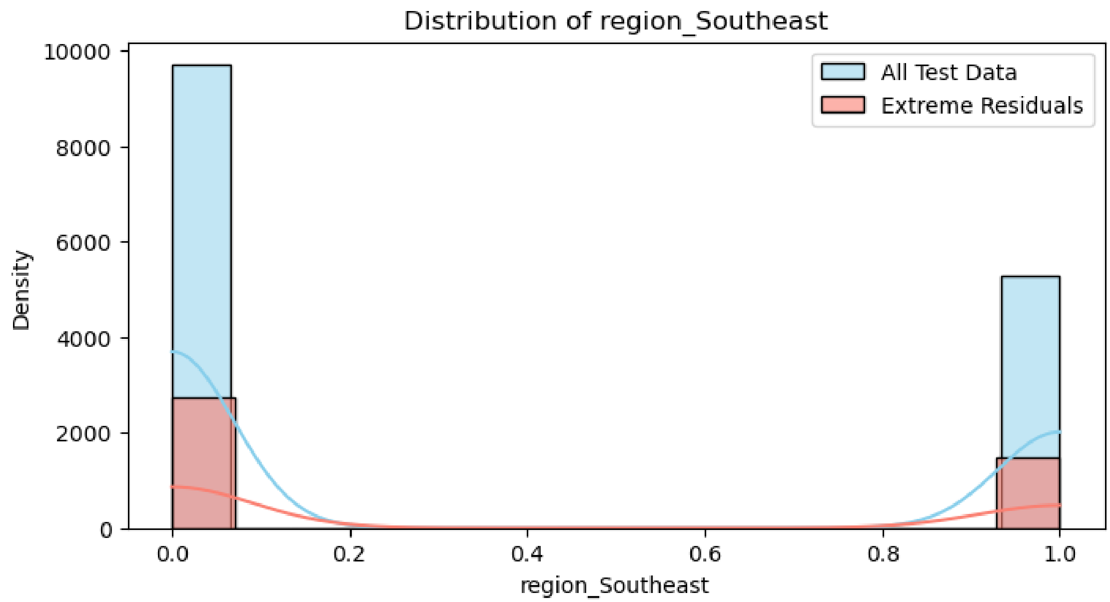
Distribution of insurance_plan
All Test Data
Extreme Residuals
Density
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
insurance_plan

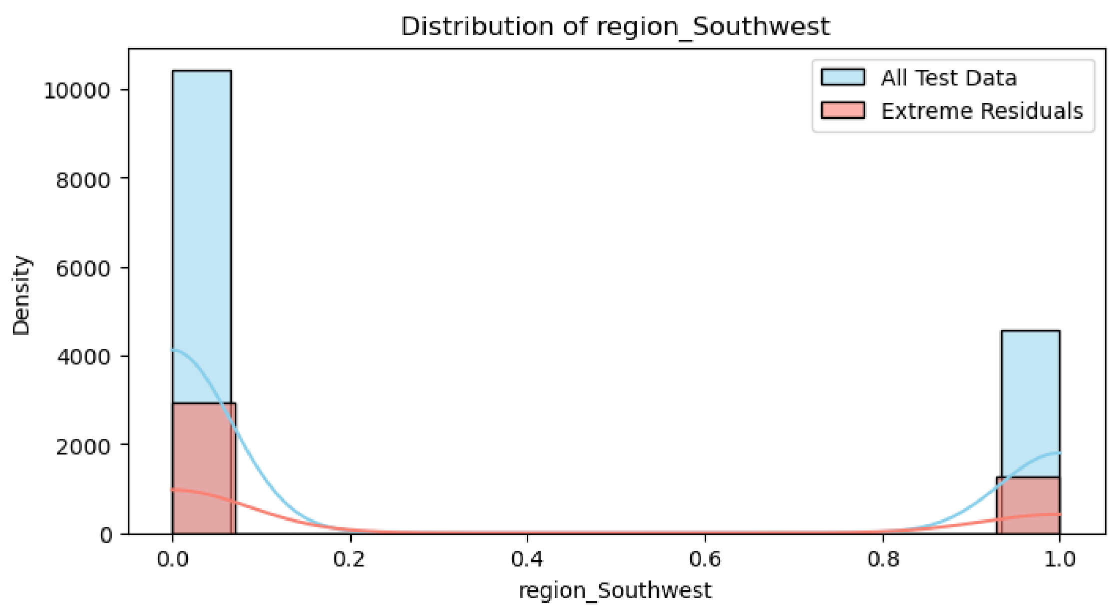


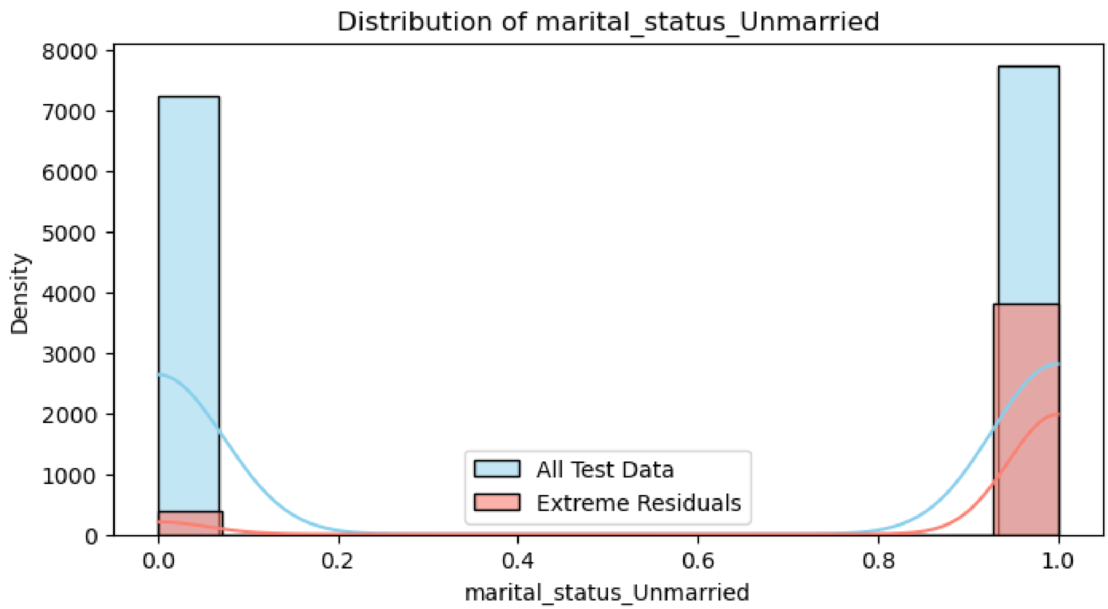
Distribution of total_risk_score
All Test Data
Extreme Residuals
Density
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
total_risk_score

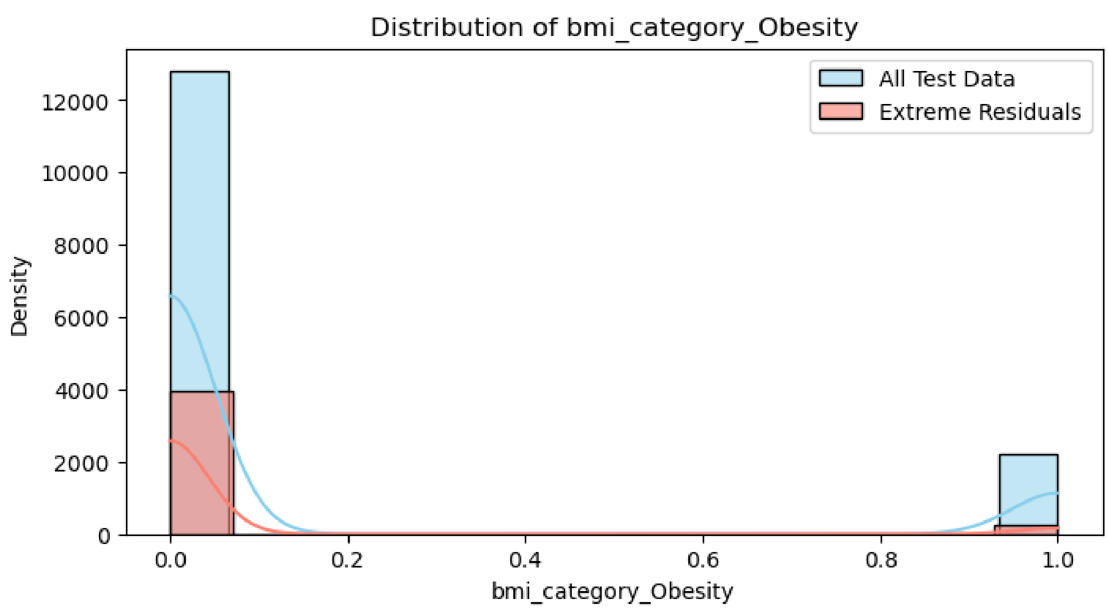

Distribution of gender_Male
All Test Data
Extreme Residuals
Density
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
gender_Male

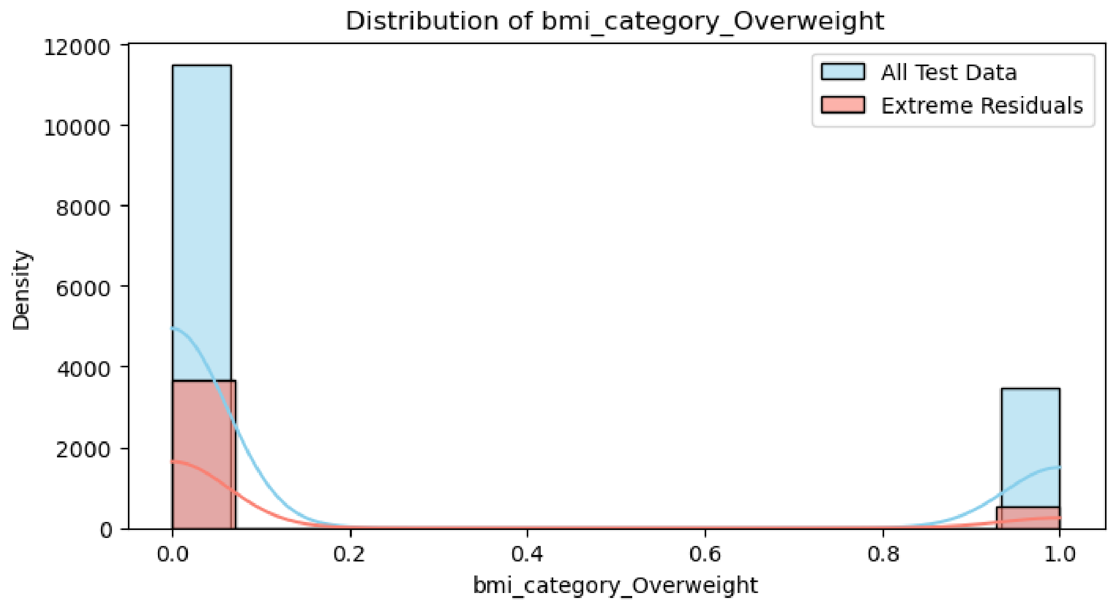

Distribution of region_Northwest
All Test Data
Extreme Residuals
Density
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
region_Northwest

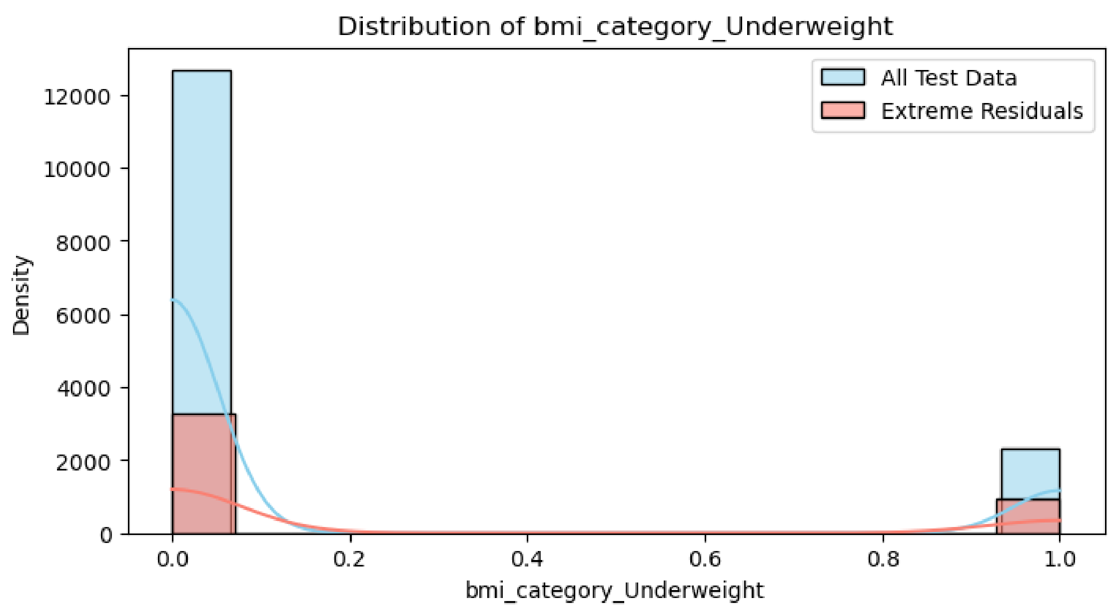

Distribution of region_Southeast
All Test Data
Extreme Residuals
Density
region_Southeast
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

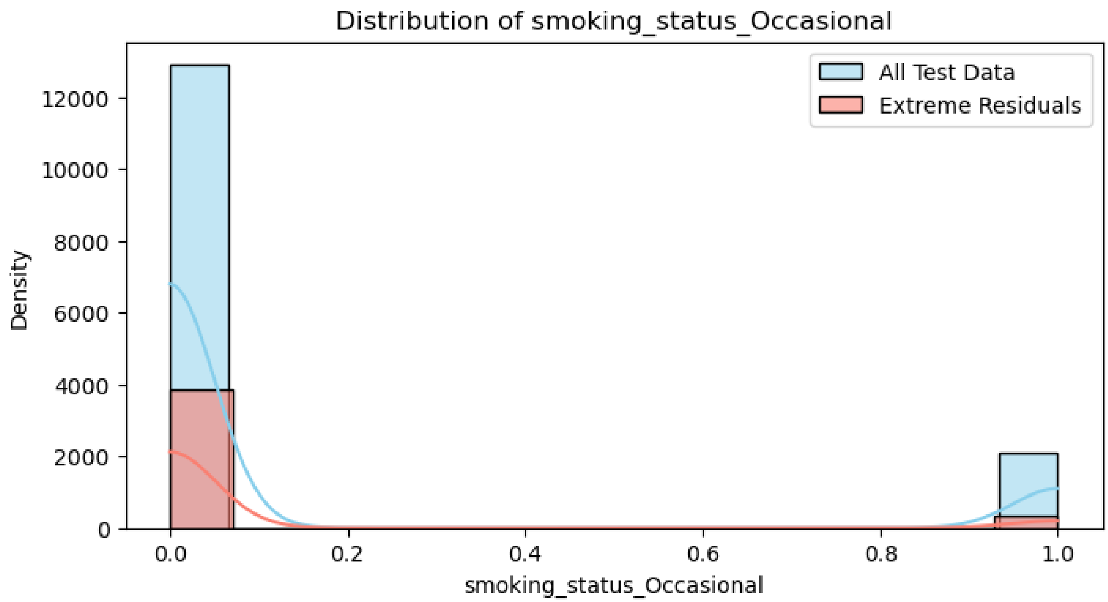

Distribution of region_Southwest
All Test Data
Extreme Residuals
Density
region_Southwest
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

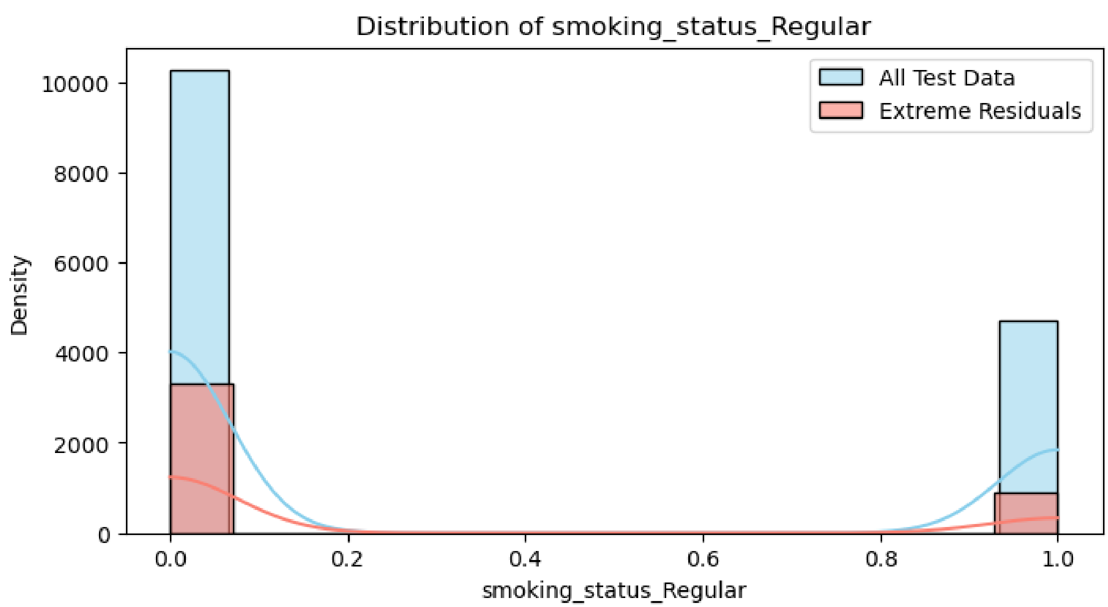

Distribution of marital_status_Unmarried
8000
7000
6000
5000
4000
3000
2000
1000
0
Density
All Test Data
Extreme Residuals
0.0
0.2
0.4
0.6
0.8
1.0
marital_status_Unmarried

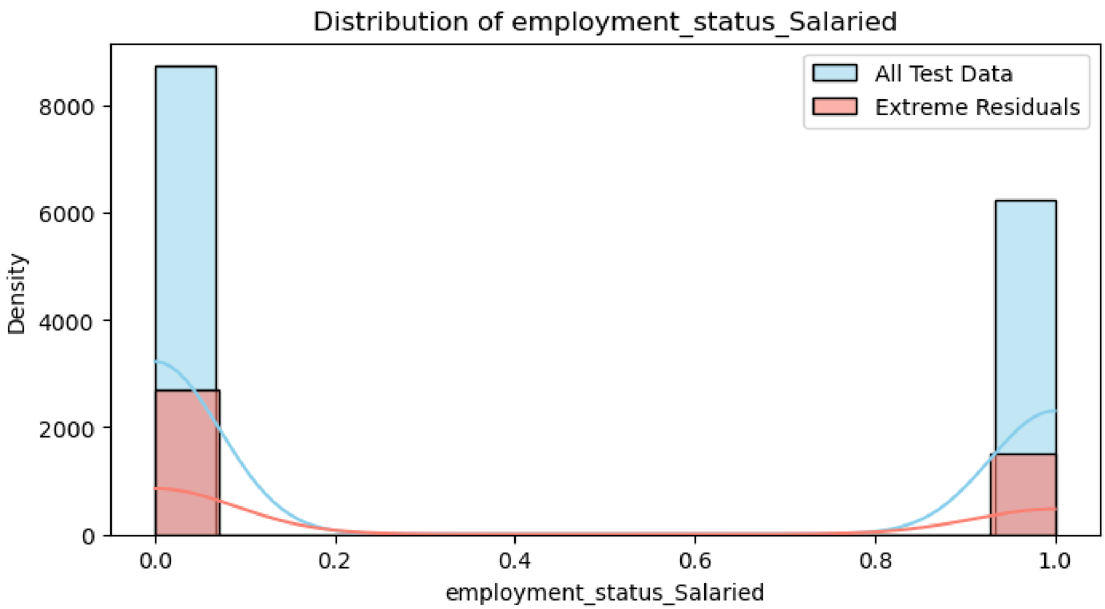

Distribution of bmi_category_Obesity
12000
10000
8000
6000
4000
2000
0
Density
All Test Data
Extreme Residuals
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Obesity

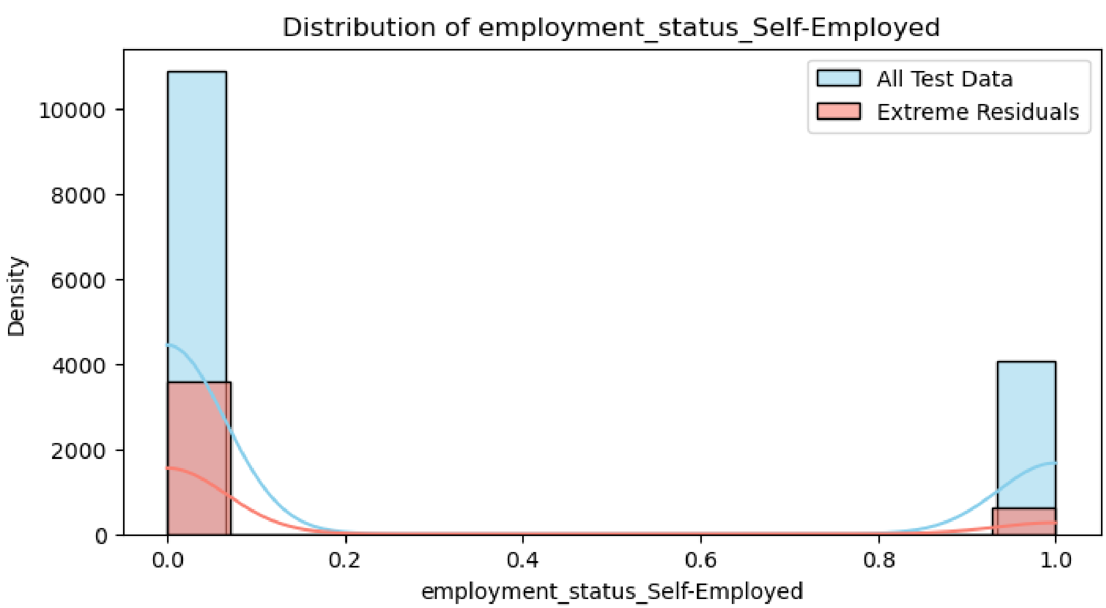

Distribution of bmi_category_Overweight
12000
10000
8000
6000
4000
2000
0
Density
All Test Data
Extreme Residuals
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Overweight


Distribution of bmi_category_Underweight
12000
10000
8000
6000
4000
2000
0
Density
All Test Data
Extreme Residuals
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Underweight


Distribution of smoking_status_Occasional
All Test Data
Extreme Residuals
Density
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
smoking_status_Occasional


Distribution of smoking_status_Regular
All Test Data
Extreme Residuals
Density
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
smoking_status_Regular





It can be observed that the distribution of the `age` feature is not aligned with the original distribution.

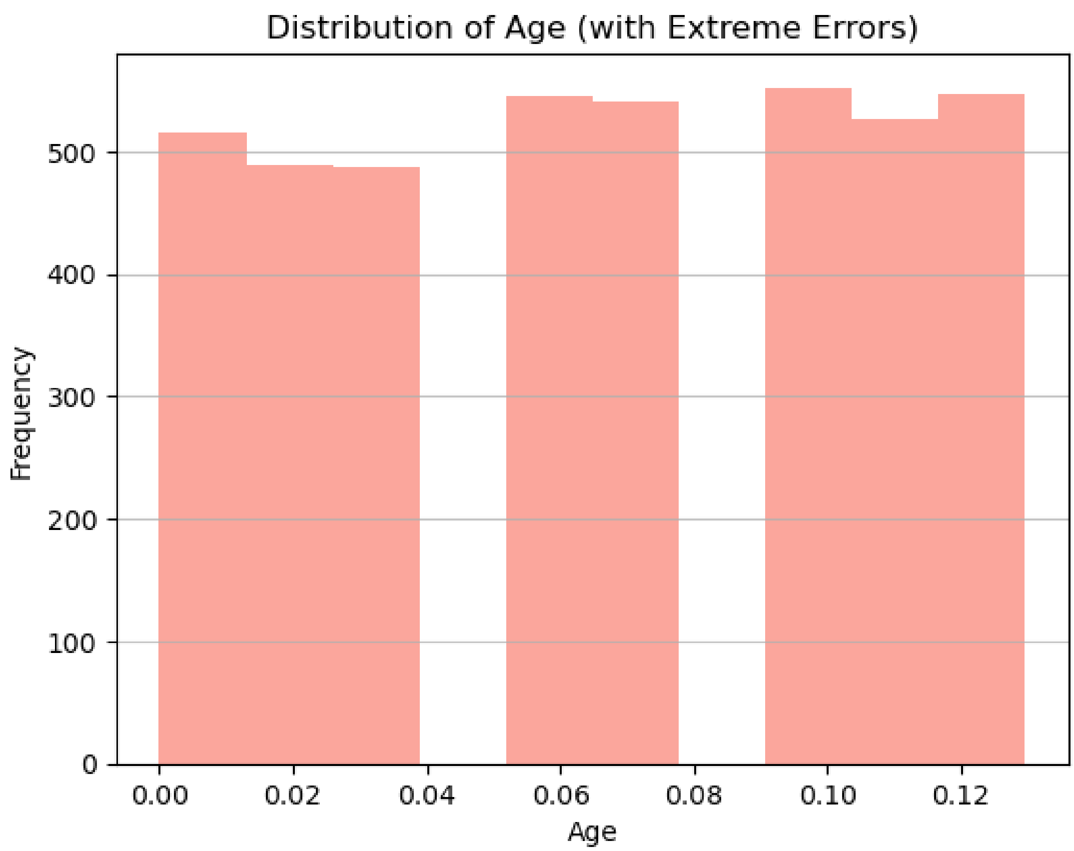
This indicates that most of the errors are clustered around the `age` feature.

0.1.3 Features Driving Extreme Residuals

We have identified that the feature `age` is associated with the extreme residuals.

Next, it is important to investigate **which specific values** of `age` feature correspond to these large residuals to better understand where the deviations occur.

[145]:
```
# Plot histogram of 'age' feature for data with extreme residuals
plt.hist(extreme_residual_df['age'], color='salmon', alpha=0.7)
plt.title('Distribution of Age (with Extreme Errors)')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



The extreme errors are observed at these specific `age` values, which are currently scaled.

To interpret them meaningfully, the scaled `age` values need to be converted back to their original scale by applying an inverse transformation.

[146]:
```
# Retrieve feature names used to fit the MinMaxScaler (mms) earlier in the␣
↪pipeline

mms.feature_names_in_
```

[146]:
```
array(['age', 'number_of_dependants', 'income_level', 'income_lakhs',
       'insurance_plan', 'total_risk_score'], dtype=object)
```

[147]:
```
# List of feature columns in the DataFrame containing extreme residual samples

extreme_residual_df.columns
```

[147]:
```
Index(['age', 'number_of_dependants', 'income_lakhs', 'insurance_plan',
       'total_risk_score', 'gender_Male', 'region_Northwest',
       'region_Southeast', 'region_Southwest', 'marital_status_Unmarried',
       'bmi_category_Obesity', 'bmi_category_Overweight',
       'bmi_category_Underweight', 'smoking_status_Occasional',
       'smoking_status_Regular', 'employment_status_Salaried',
       'employment_status_Self-Employed'],
      dtype='object')
```

The scaler object was originally trained using 6 columns:

- age
- number_of_dependants
- income_level
- income_lakhs
- insurance_plan
- total_risk_score

However, the `extreme_residual_df` contains only 5 columns:

- age
- number_of_dependants
- income_lakhs
- insurance_plan
- total_risk_score

The column `income_level` is missing from this dataframe.

[148]:
```
# Add a new dummy column 'income_level' initialized to 0 as a placeholder in␣
↪extreme_residual_df
extreme_residual_df['income_level'] = 0

# Display the first few rows to verify the new column addition
extreme_residual_df.head()
```

[148]:
```
            age  number_of_dependants  income_lakhs  insurance_plan  \
199    0.129630                   0.2      0.636364             1.0
25415  0.074074                   0.0      0.010101             0.0
47848  0.111111                   0.0      0.424242             1.0
26182  0.037037                   0.0      0.292929             0.0
16869  0.111111                   0.0      0.606061             0.5

       total_risk_score  gender_Male  region_Northwest  region_Southeast  \
199                 0.0            0                 1                 0
25415               0.0            1                 1                 0
47848               0.0            0                 1                 0
26182               0.0            1                 1                 0
16869               0.0            1                 0                 0

       region_Southwest  marital_status_Unmarried  bmi_category_Obesity  \
199                   0                         1                     0
25415                 0                         1                     0
47848                 0                         1                     0
26182                 0                         1                     0
16869                 0                         1                     0

       bmi_category_Overweight  bmi_category_Underweight  \
199                          0                         0
25415                        1                         0
47848                        0                         1
26182                        0                         1
16869                        0                         1

       smoking_status_Occasional  smoking_status_Regular  \
199                            0                       0
25415                          0                       1
47848                          0                       0
26182                          0                       0
16869                          0                       1

       employment_status_Salaried  employment_status_Self-Employed  \
199                             0                                0
25415                           1                                0
47848                           1                                0
26182                           0                                0
16869                           0                                0

       income_level
199               0
25415             0
47848             0
26182             0
```

```
16869            0
```

[149]:
```
# List of columns that were previously selected for scaling

cols_to_scale
```

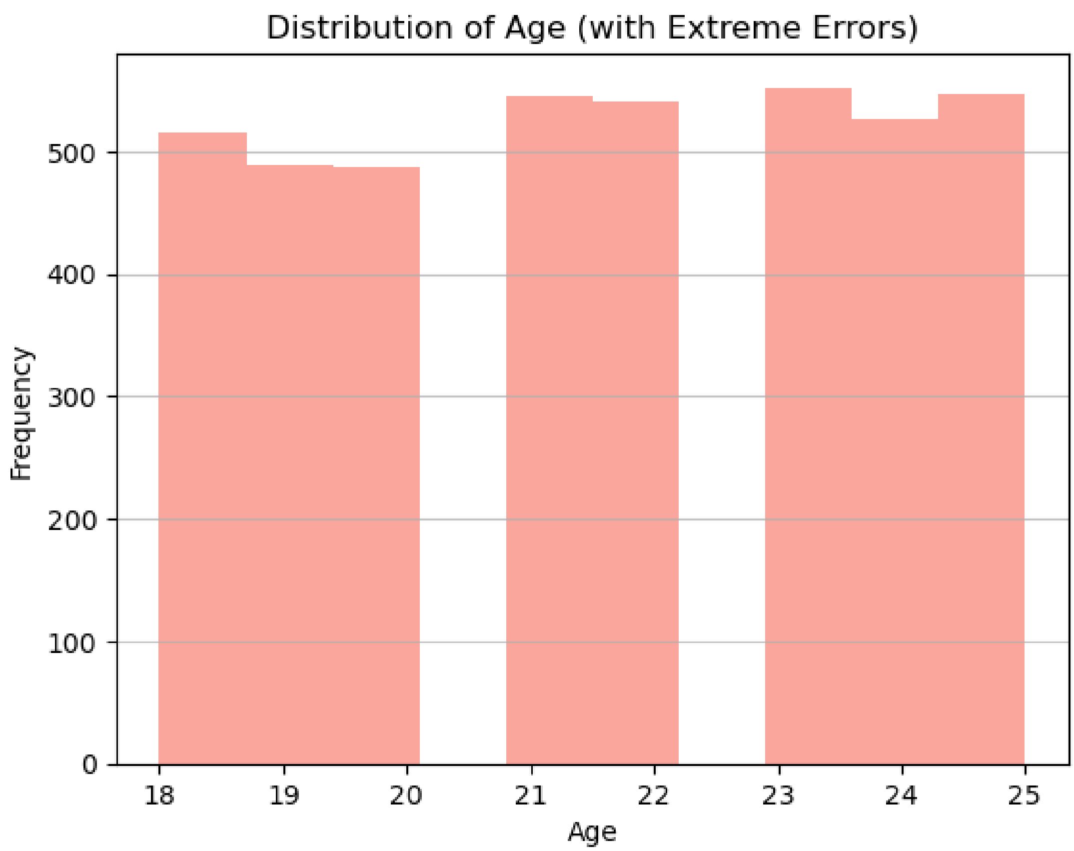
[149]:
```
['age',
 'number_of_dependants',
 'income_level',
 'income_lakhs',
 'insurance_plan',
 'total_risk_score']
```

[150]:
```
# Reverse the scaling transformation on the selected columns of␣
↪extreme_residual_df
# to get the original (descaled) feature values

descaled_extreme_residual_df = pd.DataFrame(data=mms.
↪inverse_transform(extreme_residual_df[cols_to_scale]),
                              columns=cols_to_scale)

# Display the descaled DataFrame
descaled_extreme_residual_df
```

[150]:
```
       age  number_of_dependants  income_level  income_lakhs  insurance_plan  \
0     25.0                   1.0           1.0          64.0             3.0
1     22.0                   0.0           1.0           2.0             1.0
2     24.0                   0.0           1.0          43.0             3.0
3     20.0                   0.0           1.0          30.0             1.0
4     24.0                   0.0           1.0          61.0             2.0
…      …                     …             …             …               …
4199  25.0                   2.0           1.0          25.0             1.0
4200  18.0                   1.0           1.0          19.0             1.0
4201  19.0                   2.0           1.0          18.0             1.0
4202  24.0                   1.0           1.0           3.0             1.0
4203  19.0                   0.0           1.0          32.0             1.0

      total_risk_score
0                  0.0
1                  0.0
2                  0.0
3                  0.0
4                  0.0
…                  …
4199               0.0
4200               0.0
4201               0.0
```

```
4202                0.0
4203                0.0

[4204 rows x 6 columns]
```

[151]:
```
# Plot histogram of the 'age' feature from the descaled data with extreme␣
↪residuals

plt.hist(descaled_extreme_residual_df['age'], color='salmon', alpha=0.7)
plt.title('Distribution of Age (with Extreme Errors)')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



0.1.4 Final Verdict

As observed, the age group between 18 and 25 is responsible for the most extreme errors. Therefore, the model will be segmented into two categories:

- **Young age** (25 years)

- **Rest** (> 25 years)