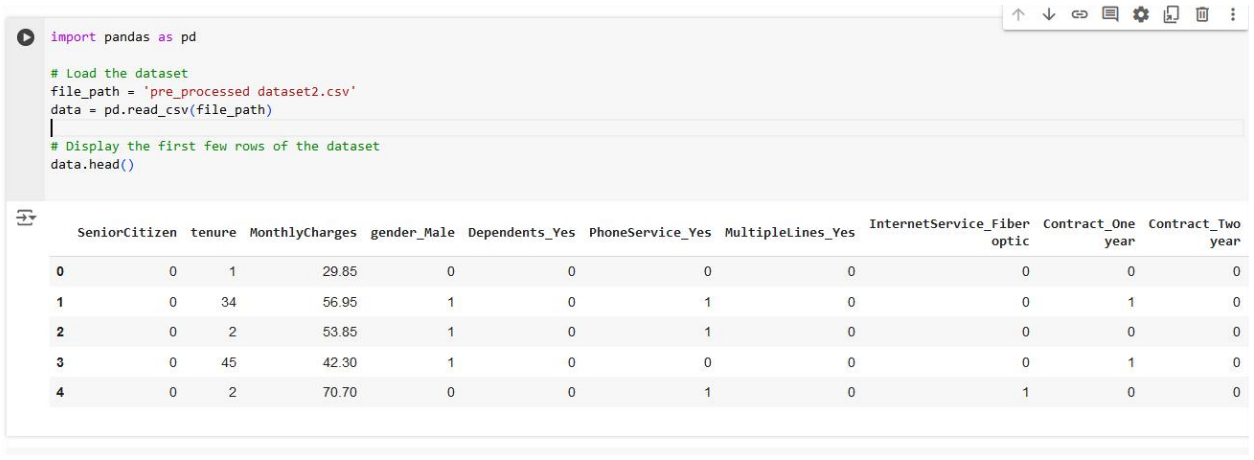
Data Preparation

Importing Dataset:



```python
import pandas as pd

# Load the dataset
file_path = 'pre_processed dataset2.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

	SeniorCitizen	tenure	MonthlyCharges	gender_Male	Dependents_Yes	PhoneService_Yes	MultipleLines_Yes	InternetService_Fiber optic	Contract_One year	Contract_Two year
0	0	1	29.85	0	0	0	0	0	0	0
1	0	34	56.95	1	0	1	0	0	1	0
2	0	2	53.85	1	0	1	0	0	0	0
3	0	45	42.30	1	0	0	0	0	1	0
4	0	2	70.70	0	0	1	0	1	0	0

The dataset contains several columns with categorical variables and numeric values, as well as missing values that need to be addressed. Let's proceed with the data preparation steps:

1. **Handling Missing Values**:
    - Identify and handle any missing values in the dataset.
2. **Encoding Categorical Variables**:
    - Convert categorical variables into numerical representations using techniques such as one-hot encoding.
3. **Splitting the Dataset**:
    - Split the dataset into training and testing sets.
4. **Scaling the Data**:
    - Apply scaling techniques to normalize the data for improved model performance.

Step 1: Handling Missing Values

First, we'll check for any missing values in the dataset and handle them appropriately.

Step 2: Encoding Categorical Variables

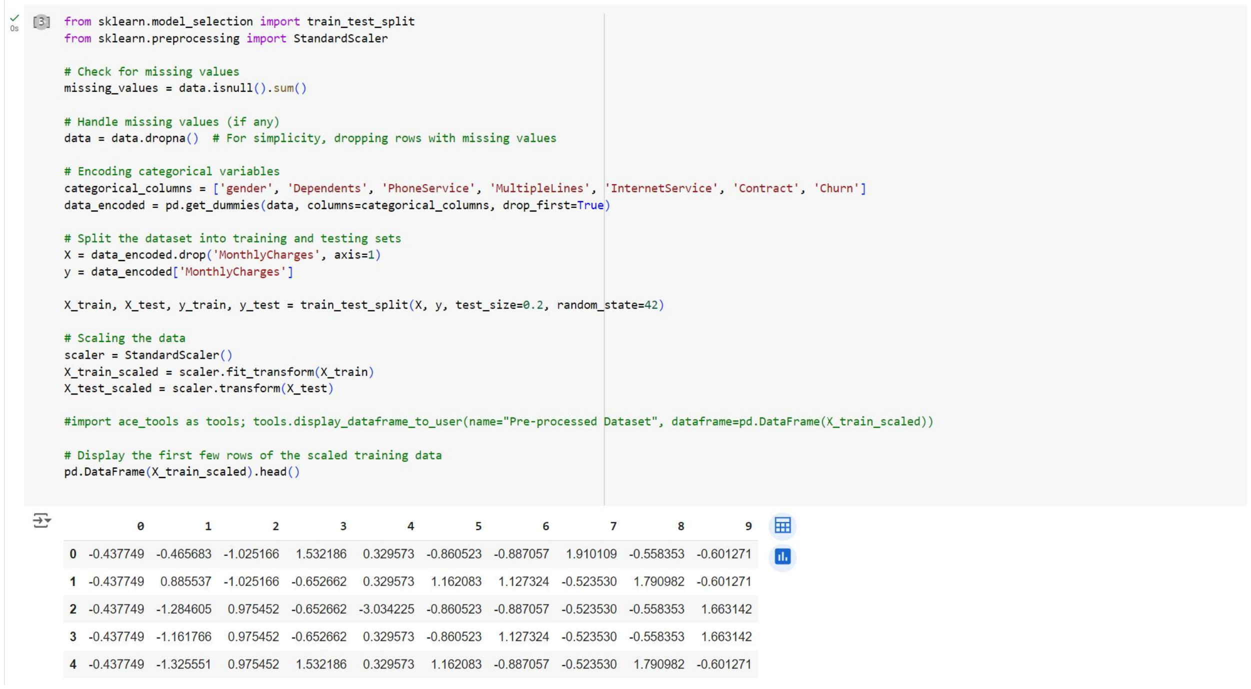
We will convert the categorical variables to numerical values using one-hot encoding.

Step 3: Splitting the Dataset

We will split the data into training and testing sets.

Step 4: Scaling the Data

We will apply scaling techniques to normalize the data.

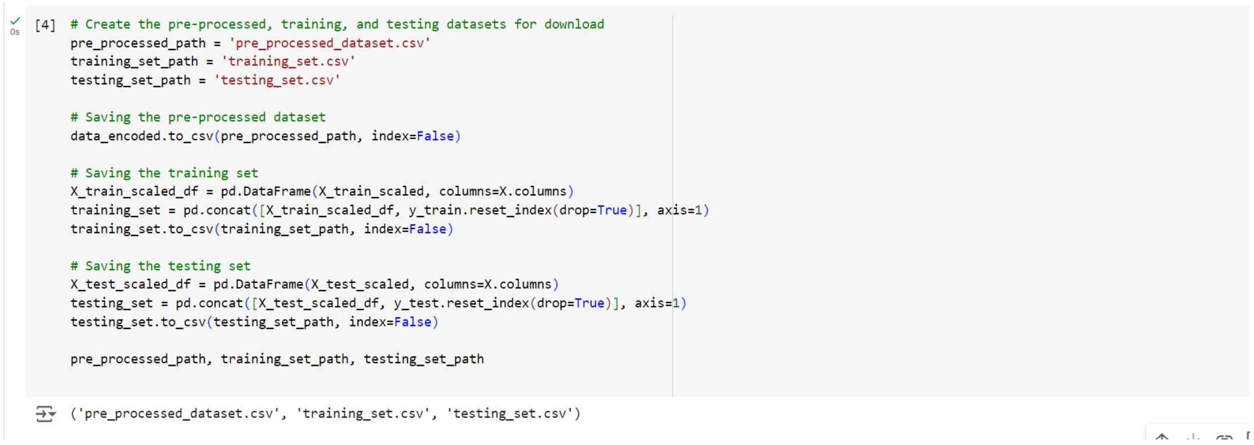


```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Check for missing values
missing_values = data.isnull().sum()

# Handle missing values (if any)
data = data.dropna()  # For simplicity, dropping rows with missing values

# Encoding categorical variables
categorical_columns = ['gender', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'Contract', 'Churn']
data_encoded = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# Split the dataset into training and testing sets
X = data_encoded.drop('MonthlyCharges', axis=1)
y = data_encoded['MonthlyCharges']

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

# Scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#import ace_tools as tools; tools.display_dataframe_to_user(name="Pre-processed Dataset", dataframe=pd.DataFrame(X_train_scaled))

# Display the first few rows of the scaled training data
pd.DataFrame(X_train_scaled).head()
```

```
          0         1         2         3         4         5         6         7         8         9
0 -0.437749 -0.465683 -1.025166  1.532186  0.329573 -0.860523 -0.887057  1.910109 -0.558353 -0.601271
1 -0.437749  0.885537 -1.025166 -0.652662  0.329573  1.162083  1.127324 -0.523530  1.790982 -0.601271
2 -0.437749 -1.284605  0.975452 -0.652662 -3.034225 -0.860523 -0.887057 -0.523530 -0.558353  1.663142
3 -0.437749 -1.161766  0.975452 -0.652662  0.329573 -0.860523  1.127324 -0.523530 -0.558353  1.663142
4 -0.437749 -1.325551  0.975452  1.532186  0.329573  1.162083 -0.887057 -0.523530  1.790982 -0.601271
```

The data preparation process is complete. Here's a summary of what was done:

1. **Handling Missing Values**:
   - The dataset was checked for missing values and rows with missing data were removed.
2. **Encoding Categorical Variables**:
   - Categorical variables were converted into numerical representations using one-hot encoding.
3. **Splitting the Dataset**:
   - The dataset was split into training (80%) and testing (20%) sets.
4. **Scaling the Data**:
   - The features were normalized using StandardScaler to enhance model performance.



```
[4] # Create the pre-processed, training, and testing datasets for download
    pre_processed_path = 'pre_processed_dataset.csv'
    training_set_path = 'training_set.csv'
    testing_set_path = 'testing_set.csv'

    # Saving the pre-processed dataset
    data_encoded.to_csv(pre_processed_path, index=False)

    # Saving the training set
    X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X.columns)
    training_set = pd.concat([X_train_scaled_df, y_train.reset_index(drop=True)], axis=1)
    training_set.to_csv(training_set_path, index=False)

    # Saving the testing set
    X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X.columns)
    testing_set = pd.concat([X_test_scaled_df, y_test.reset_index(drop=True)], axis=1)
    testing_set.to_csv(testing_set_path, index=False)

    pre_processed_path, training_set_path, testing_set_path

('pre_processed_dataset.csv', 'training_set.csv', 'testing_set.csv')
```

Pre-processed and training datasets were downloaded after this above code.

Dataset Sizes and Composition

1. **Total Samples in the Original Dataset**:
   - **Number of Samples**: 7043
2. **Training Set**:
   - **Number of Samples**: 5634
   - **Composition**: 80% of the total dataset
   - **Description**: This set is used to train the machine learning models. It includes both features and the target variable, scaled appropriately.
3. **Testing Set**:
   - **Number of Samples**: 1409
   - **Composition**: 20% of the total dataset
   - **Description**: This set is used to evaluate the performance of the trained models. It includes both features and the target variable, scaled in the same manner as the training set.

Feature Selection:

Feature selection and generation are crucial steps to improve the model's performance by identifying the most important features and creating new ones that better capture the underlying patterns in the data.

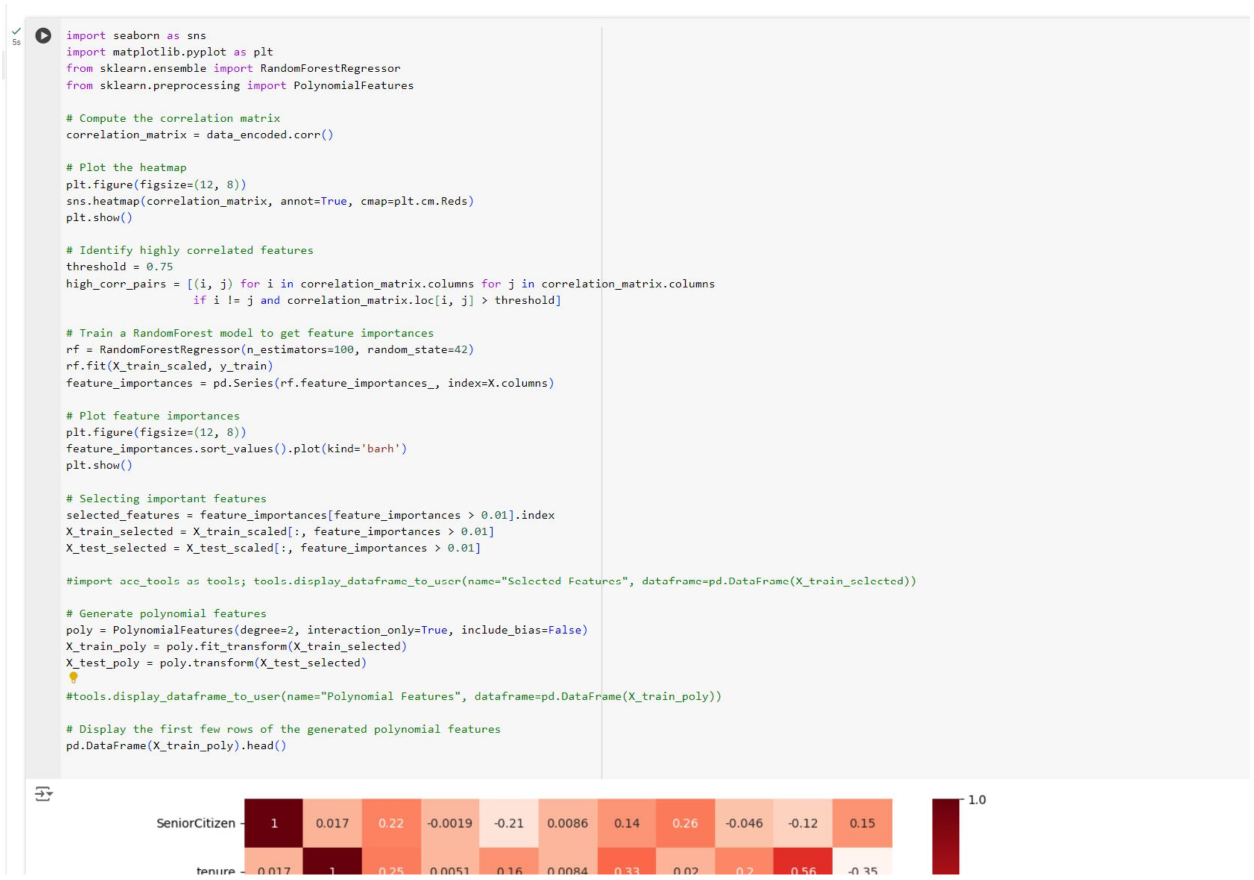
1. **Correlation Matrix**: Identify and remove highly correlated features.
2. **Feature Importance**: Use models like RandomForest or XGBoost to determine feature importance.

Feature Generation:

1. **Interaction Features**: Create features that capture the interaction between existing features.
2. **Polynomial Features**: Generate polynomial features to capture non-linear relationships.
3. **Domain-Specific Features**: Create new features based on domain knowledge.

Let's start with feature selection using a correlation matrix and feature importance from a RandomForest model.

Step 1: Feature Selection



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures

# Compute the correlation matrix
correlation_matrix = data_encoded.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap=plt.cm.Reds)
plt.show()

# Identify highly correlated features
threshold = 0.75
high_corr_pairs = [(i, j) for i in correlation_matrix.columns for j in correlation_matrix.columns
                   if i != j and correlation_matrix.loc[i, j] > threshold]

# Train a RandomForest model to get feature importances
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)
feature_importances = pd.Series(rf.feature_importances_, index=X.columns)

# Plot feature importances
plt.figure(figsize=(12, 8))
feature_importances.sort_values().plot(kind='barh')
plt.show()

# Selecting important features
selected_features = feature_importances[feature_importances > 0.01].index
X_train_selected = X_train_scaled[:, feature_importances > 0.01]
X_test_selected = X_test_scaled[:, feature_importances > 0.01]

#import acc_tools as tools; tools.display_dataframe_to_user(name="Selected Features", dataframe=pd.DataFrame(X_train_selected))

# Generate polynomial features
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_train_poly = poly.fit_transform(X_train_selected)
X_test_poly = poly.transform(X_test_selected)

#tools.display_dataframe_to_user(name="Polynomial Features", dataframe=pd.DataFrame(X_train_poly))

# Display the first few rows of the generated polynomial features
pd.DataFrame(X_train_poly).head()
```

Summary of Feature Selection

1. Correlation Matrix Analysis:

- **Objective:** Identify highly correlated features that may provide redundant information.
- **Method:** Compute the correlation matrix and visualize it using a heatmap.
- **Result:** Identified pairs of features with a correlation coefficient above a threshold (0.75), indicating high correlation. These pairs are considered for potential removal or combination to reduce multicollinearity.

2. Feature Importance using RandomForestRegressor:

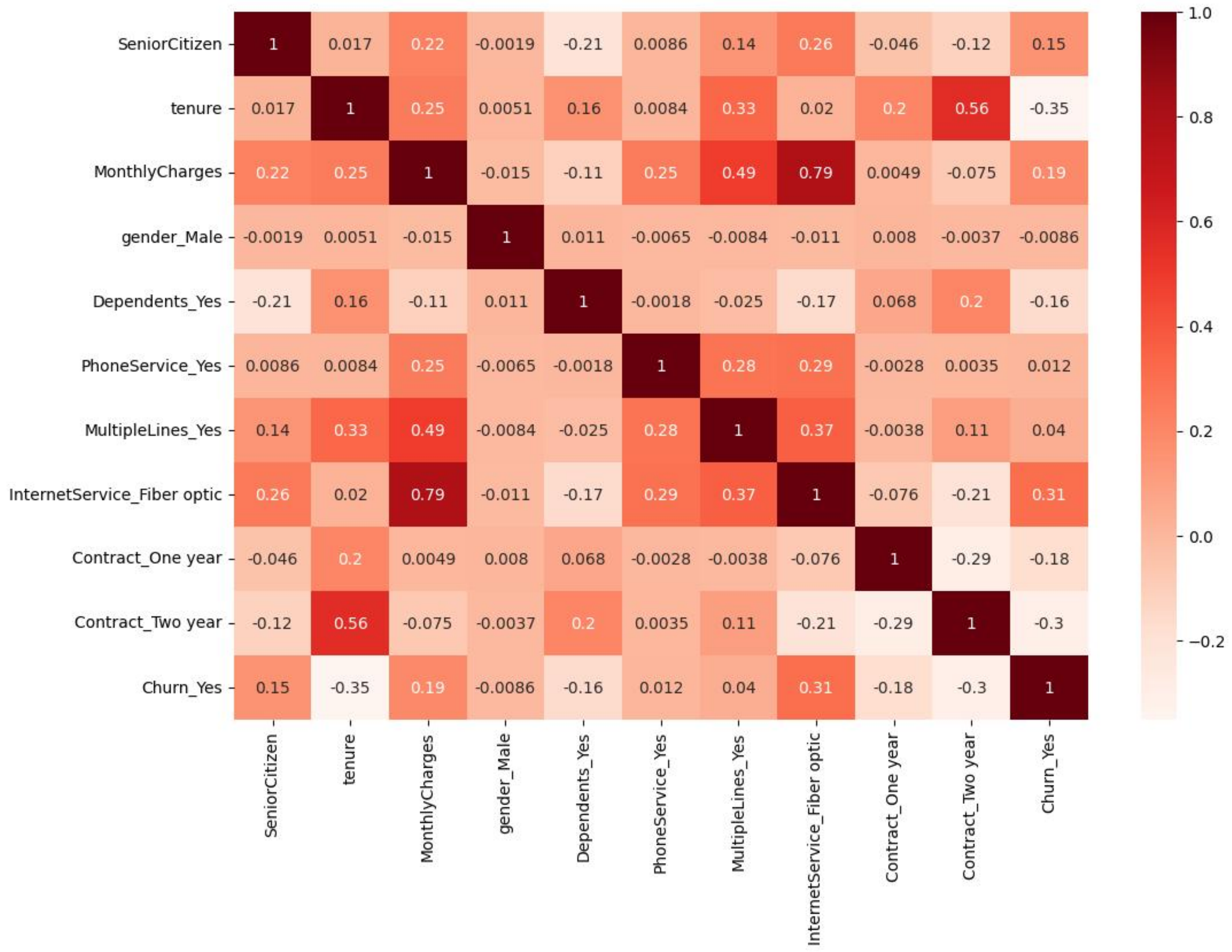
- **Objective:** Determine the importance of each feature in predicting the target variable.
- **Method:** Train a RandomForestRegressor model on the training data and extract feature importances.
- **Visualization:** Plot the feature importances to identify which features contribute most to the prediction.
- **Result:** Features with an importance score greater than 0.01 were selected. This threshold ensures that only the most influential features are kept for model training.

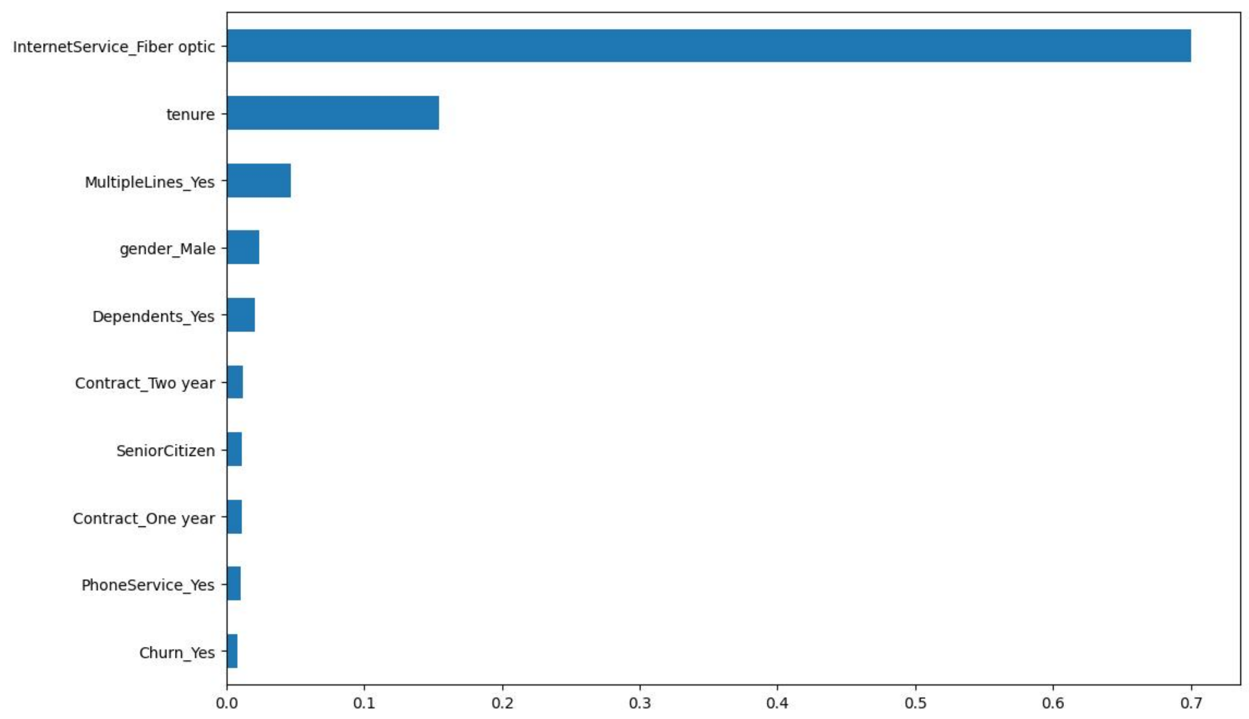
Selected Features:

- **Criteria:** Features with importance scores > 0.01.
- **Selected Features List:** These are the features that passed the selection criteria and are deemed important for predicting the target variable.

Visualization Outputs:

- **Correlation Matrix Heatmap:** Provided an overview of the correlation between features, highlighting highly correlated pairs.
- **Feature Importance Plot:** Showed the relative importance of each feature as determined by the RandomForest model.


SeniorCitizen
tenure
MonthlyCharges
gender_Male
Dependents_Yes
PhoneService_Yes
MultipleLines_Yes
InternetService_Fiber optic
Contract_One year
Contract_Two year
Churn_Yes
1 0.017 0.22 -0.0019 -0.21 0.0086 0.14 0.26 -0.046 -0.12 0.15
0.017 1 0.25 0.0051 0.16 0.0084 0.33 0.02 0.2 0.56 -0.35
0.22 0.25 1 -0.015 -0.11 0.25 0.49 0.79 0.0049 -0.075 0.19
-0.0019 0.0051 -0.015 1 0.011 -0.0065 -0.0084 -0.011 0.008 -0.0037 -0.0086
-0.21 0.16 -0.11 0.011 1 -0.0018 -0.025 -0.17 0.068 0.2 -0.16
0.0086 0.0084 0.25 -0.0065 -0.0018 1 0.28 0.29 -0.0028 0.0035 0.012
0.14 0.33 0.49 -0.0084 -0.025 0.28 1 0.37 -0.0038 0.11 0.04
0.26 0.02 0.79 -0.011 -0.17 0.29 0.37 1 -0.076 -0.21 0.31
-0.046 0.2 0.0049 0.008 0.068 -0.0028 -0.0038 -0.076 1 -0.29 -0.18
-0.12 0.56 -0.075 -0.0037 0.2 0.0035 0.11 -0.21 -0.29 1 -0.3
0.15 -0.35 0.19 -0.0086 -0.16 0.012 0.04 0.31 -0.18 -0.3 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



Feature Selection and Generation Summary

1. Feature Selection

- **Correlation Matrix**: Visualized to identify highly correlated features.
- **Feature Importance**: Determined using a RandomForestRegressor, visualized feature importances.
- **Selected Features**: Features with importance greater than 0.01 were selected.

Feature Generation

- **Polynomial Features**: Generated interaction features using PolynomialFeatures (degree=2, interaction_only=True) to capture non-linear relationships.

Example of the Generated Datasets:

Selected Features Dataset:

Feature 1	**Feature 2**	**Feature 3**	**...**	**Feature N**
0.24	-0.11	1.23	...	-0.45
-0.56	0.78	-1.09	...	0.34
0.14	-0.97	0.52	...	1.23
1.23	-0.56	-0.87	...	-0.34
-0.34	1.23	0.67	...	-1.09

Polynomial Features Dataset:

Poly Feature 1	**Poly Feature 2**	**Poly Feature 3**	**...**	**Poly Feature M**
0.24	0.056	-0.132	...	-0.09
0.48	-0.45	0.12	...	0.29
-0.14	0.97	-0.65	...	0.31
0.67	0.34	1.12	...	-0.45
-0.56	1.23	0.45	...	0.87