CS 5805 - MACHINE LEARNING I

FINAL TERM PROJECT

AT&T INTERNET SPEEDS AND PRICES ANALYSIS

Presented by -

Mihir Rathod

ABSTRACT

This project analyzes the AT&T dataset on internet speeds, prices, locations, and socioeconomic factors to explore how regional differences impact broadband availability, affordability, and quality in the United States. It highlights how disparities and market dynamics shape internet services across different areas.

Through this work, I learned the value of feature selection techniques like Variance Inflation Factor (VIF) to reduce multicollinearity and improve model reliability. Comparing various machine learning algorithms helped identify the most effective ones for specific tasks. Clustering and association rule mining offered deeper insights into customer behavior and service trends, showing the benefits of combining supervised and unsupervised learning methods.

The research is divided into four phases. Phase I focuses on feature engineering and exploratory data analysis (EDA) to refine prediction accuracy. Phase II uses regression analysis on upload speeds to identify key predictors, like download speeds and broadband usage, and their impact on performance. Phase III compares seven classification models, including Logistic Regression, Neural Networks, and Decision Trees, to predict internet package types. Logistic Regression stood out for its simplicity, efficiency, and high performance. Phase IV uses clustering models, like K-Means and DBSCAN, and association rule mining to uncover patterns and relationships between features.

These findings aim to help businesses, researchers, and other stakeholders close the digital divide, improve internet access, and promote digital equity across the country.

CONTENTS

List Of Figures

INTRODUCTION

This dataset looks at how internet services and local factors connect across the United States. It focuses on internet speeds, prices, and broadband availability, showing how different areas experience access and affordability. It's a useful tool for policymakers, businesses, and researchers working to improve digital access.

The data includes details like download and upload speeds, service prices, and types of internet packages. It also covers socio-economic and geographic information such as population density, broadband subscription rates, median household income, and the number of competitors in each area.

By combining these details, the dataset helps explain how factors like income and population affect internet access. It raises questions like whether areas with more competition or higher broadband use get faster speeds or better prices. It also highlights underserved areas with limited or expensive internet services.

This dataset can help businesses expand broadband and understand market opportunities. It's a valuable resource for promoting fair and affordable internet access.

Description of the dataset

The dataset originally consists of 432,303 observations and 26 features, with the following breakdown:

- 11 Categorical Features
- 15 Numerical Features

Numerical Features:

These include Download Speeds, Upload Speeds, Latitude, Longitude, Number of Providers, and other related variables.

Categorical Features:

These include State, Technology, Major City, Package Type, and other categorical variables.

Target Variables:

Phase I & II (Regression): 'speed_up' – Upload Speeds.

Phase III (Classification): 'package' – Type of internet package, with 4 classes.

Importance of the dataset

The AT&T Internet Speeds, Prices, and Socioeconomic Dataset helps uncover differences in internet access, pricing, and service quality across the U.S. It shows how factors like competition, population density, and income impact broadband availability and affordability. This dataset is a valuable tool for businesses and researchers to make smarter decisions about market competition, pricing, and infrastructure. It also supports efforts to close the digital divide, expand access to high-speed internet, and boost economic growth through better connectivity.

Objectives

Phase I: Perform feature engineering and exploratory data analysis (EDA) to understand the dataset, implement different models for feature selection, and choose the best feature selection model for further implementation.

Phase II: Implement regression on a continuous feature (‘speed_up’) to predict and analyze the upload speed using other predictors in the dataset.

Phase III: Implement seven different classifiers on the categorical target variable (‘package’), analyze the results, and pick the best classifier based on their performances and other metrics.

Phase IV: Implement clustering using two clustering models, perform association rule mining using Apriori algorithm model, and analyze the results.

PHASE I: FEATURE ENGINEERING & EDA

Head of the Original Dataset

Loading and printing the head of original dataset -

```
Head of the dataset:
                               address_full incorporated_place  \
0  2406 Country Club Ave NW, Huntsville AL 35816    Huntsville city
1       1902 Oglesby Dr NW, Huntsville AL 35816    Huntsville city
2     2312 Cardinal Ave NW, Huntsville AL 35816    Huntsville city
3       1903 Oglesby Dr NW, Huntsville AL 35816    Huntsville city
4   1905 Canterbury Cir NW, Huntsville AL 35816    Huntsville city

   major_city state    lat     lon  block_group  collection_datetime provider  \
0  huntsville    AL 34.745 -86.607  10890007022           1650310200     AT&T
1  huntsville    AL 34.748 -86.607  10890007022           1650310229     AT&T
2  huntsville    AL 34.747 -86.606  10890007022           1650310203     AT&T
3  huntsville    AL 34.748 -86.608  10890007022           1650310195     AT&T
4  huntsville    AL 34.749 -86.605  10890007022           1650310196     AT&T

   speed_down  speed_up speed_unit  price technology                  package  \
0       0.768     0.384       Mbps 55.000  Not Fiber   Internet Basic 768kbps
1       5.000     1.000       Mbps 55.000  Not Fiber         Internet Basic 5
2       0.768     0.384       Mbps 55.000  Not Fiber   Internet Basic 768kbps
3       5.000     1.000       Mbps 55.000  Not Fiber         Internet Basic 5
4     300.000   300.000       Mbps 55.000      Fiber  AT&T FIBER—INTERNET 300

   fastest_speed_down  fastest_speed_price  \
0               0.768               55.000
1               5.000               55.000
2               0.768               55.000
3               5.000               55.000
4           5,000.000              180.000
```

```
                                                              fn  \
0  ../data/intermediary/isp/att/huntsville/010890007022.geo...
1  ../data/intermediary/isp/att/huntsville/010890007022.geo...
2  ../data/intermediary/isp/att/huntsville/010890007022.geo...
3  ../data/intermediary/isp/att/huntsville/010890007022.geo...
4  ../data/intermediary/isp/att/huntsville/010890007022.geo...

  redlining_grade  race_perc_non_white  income_lmi  ppl_per_sq_mile  \
0             NaN                0.475       0.382          512.090
1             NaN                0.475       0.382          512.090
2             NaN                0.475       0.382          512.090
3             NaN                0.475       0.382          512.090
4             NaN                0.475       0.382          512.090

   n_providers  income_dollars_below_median  internet_perc_broadband  \
0        4.000                   35,091.000                    0.528
1        4.000                   35,091.000                    0.528
2        4.000                   35,091.000                    0.528
3        4.000                   35,091.000                    0.528
4        4.000                   35,091.000                    0.528

   median_household_income
0                    21667
1                    21667
2                    21667
3                    21667
4                    21667
```

Data Cleaning

These are the number of missing observations in my dataset -

```
The no. of missing observations in the dataset before cleaning:
address_full                        0
incorporated_place                  0
major_city                          0
state                               0
lat                                 0
lon                                 0
block_group                         0
collection_datetime                 0
provider                            0
speed_down                          0
speed_up                            0
speed_unit                      82600
price                           82600
technology                      82600
package                         82600
fastest_speed_down                  0
fastest_speed_price                 0
fn                                  0
redlining_grade                246027
race_perc_non_white                 0
income_lmi                      17661
ppl_per_sq_mile                  7965
n_providers                      7969
income_dollars_below_median     17661
internet_perc_broadband           849
median_household_income             0
dtype: int64
```

Handled the missing values by performing these steps –

- Dropped the unnecessary features such as - 'collection_datetime', 'fn', 'address_full', 'incorporated_place', 'major_city', 'provider', 'speed_unit', 'income_lmi', 'income_dollars_below_median'.
- Dropped nan values for 'price', 'technology', 'package'.
- Merged similar package names in ‘package’ for simplicity.
- Filled the nan values using mode for ‘redlining_grade’ and ‘n_providers’.
- Filled the nan values using median for ‘ppl_per_sq_mile’ and ‘internet_perc_broadband’.

These are the number of missing observations (after cleaning) -

```
The number of missing observations in the dataset after cleaning:
state                      0
lat                        0
lon                        0
block_group                0
speed_down                 0
speed_up                   0
price                      0
technology                 0
package                    0
fastest_speed_down         0
fastest_speed_price        0
redlining_grade            0
race_perc_non_white        0
ppl_per_sq_mile            0
n_providers                0
internet_perc_broadband    0
median_household_income    0
dtype: int64
```

Head of the cleaned dataset

After handling the missing values, here is the head of the cleaned dataset -

```
Displaying the cleaned dataset:
  state    lat     lon  block_group  speed_down  speed_up  price technology  \
0    AL 34.745 -86.607  10890007022       0.768     0.384 55.000  Not Fiber
1    AL 34.748 -86.607  10890007022       5.000     1.000 55.000  Not Fiber
2    AL 34.747 -86.606  10890007022       0.768     0.384 55.000  Not Fiber
3    AL 34.748 -86.608  10890007022       5.000     1.000 55.000  Not Fiber
4    AL 34.749 -86.605  10890007022     300.000   300.000 55.000      Fiber

          package  fastest_speed_down  fastest_speed_price redlining_grade  \
0  Basic Internet               0.768               55.000               C
1  Basic Internet               5.000               55.000               C
2  Basic Internet               0.768               55.000               C
3  Basic Internet               5.000               55.000               C
4  Fiber Internet           5,000.000              180.000               C

   race_perc_non_white  ppl_per_sq_mile  n_providers  internet_perc_broadband  \
0                0.475          512.090        4.000                    0.528
1                0.475          512.090        4.000                    0.528
2                0.475          512.090        4.000                    0.528
3                0.475          512.090        4.000                    0.528
4                0.475          512.090        4.000                    0.528

   median_household_income
0                    21667
1                    21667
2                    21667
3                    21667
4                    21667
```

Checking duplicates

There were 1875 duplicates in the dataset. Dropped all the duplicates from the existing dataset.

```
*****************************************************************

Checking whether the dataset has any duplications (before): 1875

*****************************************************************

Checking whether the dataset has any duplications (after): 0

*****************************************************************
```

Data Aggregation

I chose to aggregate the dataset as there was minimal variability within certain features. Aggregating by census block groups helped the dataset to be more varied which made it suitable for regression and classification tasks in Phases 2 and 3.

Grouping by 'block_group'

- The subset of the dataset is grouped by 'block_group' – it represents a census block group based on the latitude and longitude.

Aggregation

Using mean:

- 'price', 'speed_down', 'speed_up', 'ppl_per_sq_mile', 'internet_perc_broadband', 'lat', 'lon', 'race_perc_non_white' and 'median_household_income'.

Using mode:

- 'package'.

Using sum:

- 'n_providers'.

Aggregated Dataset -

```
I am going to use this aggregated_att in phase 2 and 3:
   block_group  price  speed_down  speed_up  n_providers  ppl_per_sq_mile  \
0  10890002021 55.000     289.464   289.321      112.000          696.213
1  10890002022 55.000     300.000   300.000      145.000          619.672
2  10890003011 54.554     294.821   294.661      224.000          990.162
3  10890003012 55.000     283.838   283.838      148.000          907.839
4  10890003013 55.000     288.031   288.015      125.000        1,495.346

   internet_perc_broadband    lat     lon  race_perc_non_white  \
0                    0.586 34.769 -86.580                0.890
1                    0.618 34.766 -86.581                0.740
2                    0.808 34.781 -86.588                0.719
3                    0.799 34.788 -86.586                0.888
4                    0.752 34.790 -86.595                0.955

   median_household_income         package
0               15,992.000  Fiber Internet
1               26,094.000  Fiber Internet
2               36,964.000  Fiber Internet
3               24,850.000  Fiber Internet
4               34,167.000  Fiber Internet
```

Data downsampling

I downsampled the dataset to 30% of its original size to reduce computational overhead and enable faster processing during Phase 4 clustering. This ensures efficient analysis while maintaining the diversity and representativeness of the data.

Shape of Downsampled dataset -

```
**********************************************************************

I am using the downsampled dataset in phase 4 for faster computation.
The shape of the downsampled dataset is (104348, 8)

**********************************************************************
```

Performing One-Hot Encoding

Now, performing One-Hot Encoding, avoiding the dummy trap by using get_dummies().

Implementing it on the categorical features -
'state', 'package', 'technology', 'redlining_grade'.

This is the head of the encoded dataset -

```
Displaying the encoded dataset-
     lat     lon  block_group  speed_down  speed_up  price  \
0 34.745 -86.607  10890007022       0.768     0.384 55.000
1 34.748 -86.607  10890007022       5.000     1.000 55.000
2 34.747 -86.606  10890007022       0.768     0.384 55.000
3 34.748 -86.608  10890007022       5.000     1.000 55.000
4 34.749 -86.605  10890007022     300.000   300.000 55.000

   fastest_speed_down  fastest_speed_price  race_perc_non_white  \
0               0.768               55.000                0.475
1               5.000               55.000                0.475
2               0.768               55.000                0.475
3               5.000               55.000                0.475
4           5,000.000              180.000                0.475

   ppl_per_sq_mile  n_providers  internet_perc_broadband  \
0          512.090        4.000                    0.528
1          512.090        4.000                    0.528
2          512.090        4.000                    0.528
3          512.090        4.000                    0.528
4          512.090        4.000                    0.528

   median_household_income  state_AR  state_CA  state_FL  state_GA
0                    21667     False     False     False     False
1                    21667     False     False     False     False
2                    21667     False     False     False     False
3                    21667     False     False     False     False
4                    21667     False     False     False     False
```

```
   state_IN  state_KS  state_KY  state_LA  state_MI  state_MO  s
0     False     False     False     False     False     False
1     False     False     False     False     False     False
2     False     False     False     False     False     False
3     False     False     False     False     False     False
4     False     False     False     False     False     False

   state_NC  state_OH  state_OK  state_SC  state_TN  state_TX  s
0     False     False     False     False     False     False
1     False     False     False     False     False     False
2     False     False     False     False     False     False
3     False     False     False     False     False     False
4     False     False     False     False     False     False

   package_AT&T FIBER—INTERNET 100  package_AT&T FIBER—INTERNET :
0                            False                             F
1                            False                             F
2                            False                             F
3                            False                             F
4                            False                             F
```

```
   package_AT&T FIBER-INTERNET 500  package_Basic Internet  \
0                            False                    True
1                            False                    True
2                            False                    True
3                            False                    True
4                            False                   False

   package_Fiber Internet  package_High-Speed Internet  technology_Not Fiber
0                   False                        False                 True
1                   False                        False                 True
2                   False                        False                 True
3                   False                        False                 True
4                    True                        False                False

   redlining_grade_B  redlining_grade_C  redlining_grade_D
0              False               True              False
1              False               True              False
2              False               True              False
3              False               True              False
4              False               True              False
```

Splitting the dataset

Splitting the cleaned dataset to perform dimensionality reduction and feature selection for selecting the best features for other phases.

Target Variable:
Selected ‘speed_up’ (upload speeds) as the target variable for regression.
I chose ‘speed_up’ as it directly reflects internet service performance, which is essential for understanding service quality and optimizing internet access across different regions.

Feature Matrix and Target Variable:
X: Features (all columns except ‘speed_up’).
y: Target variable (only ‘speed_up’).

Train-Test Split:
Split the dataset into training (80%) and testing (20%) subsets.
Performed with shuffle=True to ensure randomness in the split and prevent any bias in the training and testing sets.

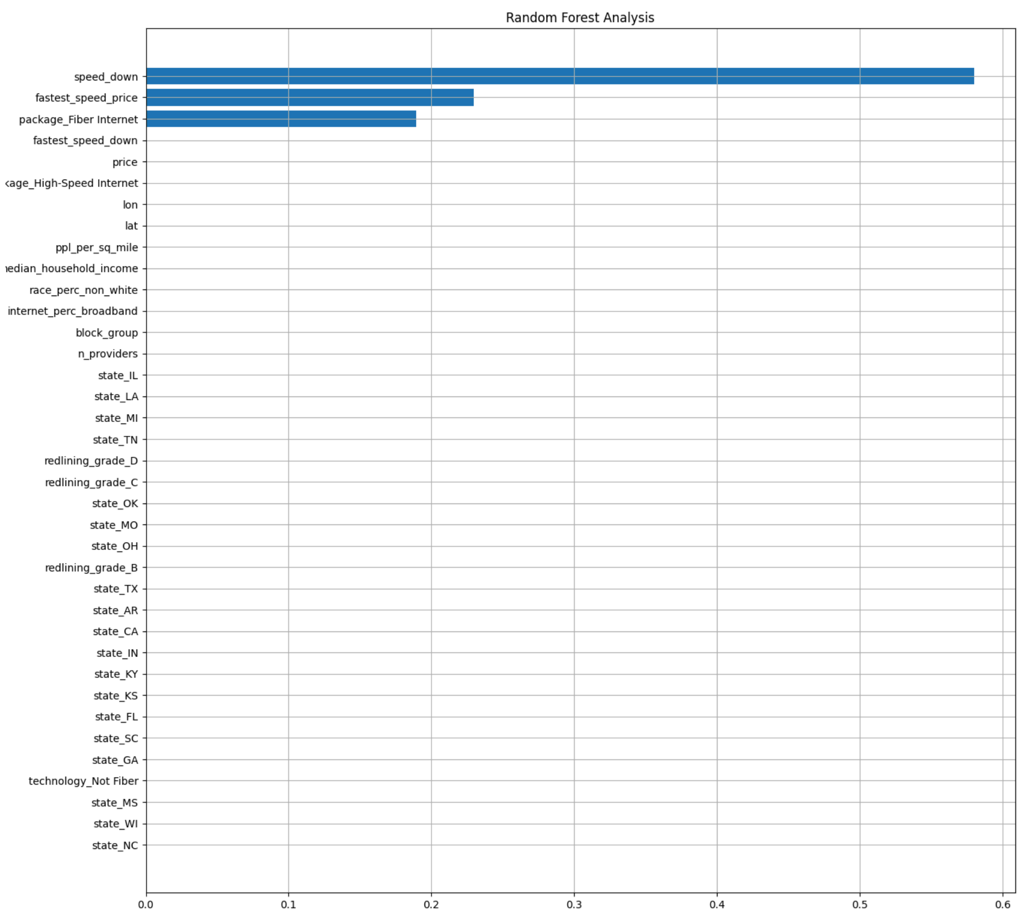
Dimensionality Reduction/Feature Selection

Dimensionality reduction and feature selection are techniques used to reduce the number of input variables in a dataset, aiming to improve model performance by focusing on the most important features. To achieve this, I applied various methods like **Random Forest**, **PCA**, **SVD**, and **VIF**, which helped me identify and retain the most relevant features while eliminating unnecessary ones.

Random Forest Analysis

Random Forest Analysis was used as a feature selection method to identify the most significant predictors for the target variable (speed_up).It is used to determine the most important features for predicting the target (‘speed_up’). This ensures that only the most impactful features are retained for further modeling.

Feature Importance Plot - Random Forest Analysis -



This graph highlights the relative importance of all the features for the target.

It is evident that ‘speed_down’,‘fastest_speed_price’ and 'package_Fiber Internet' have shown the highest importance in predicting the target.

Using a threshold of 0.01, segregated the selected features and the eliminated features according to their importances to the target variable.

Selected and Eliminated features -

```
Selected Features (Random Forest): Index(['speed_down', 'fastest_speed_price', 'package_Fiber Internet'], dtype='object')

Eliminated Features (Random Forest): Index(['fastest_speed_down', 'price', 'package_High-Speed Internet', 'lon',
       'lat', 'ppl_per_sq_mile', 'median_household_income',
       'race_perc_non_white', 'internet_perc_broadband', 'block_group',
       'n_providers', 'state_IL', 'state_LA', 'state_MI', 'state_TN',
       'redlining_grade_D', 'redlining_grade_C', 'state_OK', 'state_MO',
       'state_OH', 'redlining_grade_B', 'state_TX', 'state_AR', 'state_CA',
       'state_IN', 'state_KY', 'state_KS', 'state_FL', 'state_SC', 'state_GA',
       'technology_Not Fiber', 'state_MS', 'state_WI', 'state_NC'],
      dtype='object')
```

Selected Features are ‘speed_down’, ‘fastest_speed_price’ and 'package_Fiber Internet'.

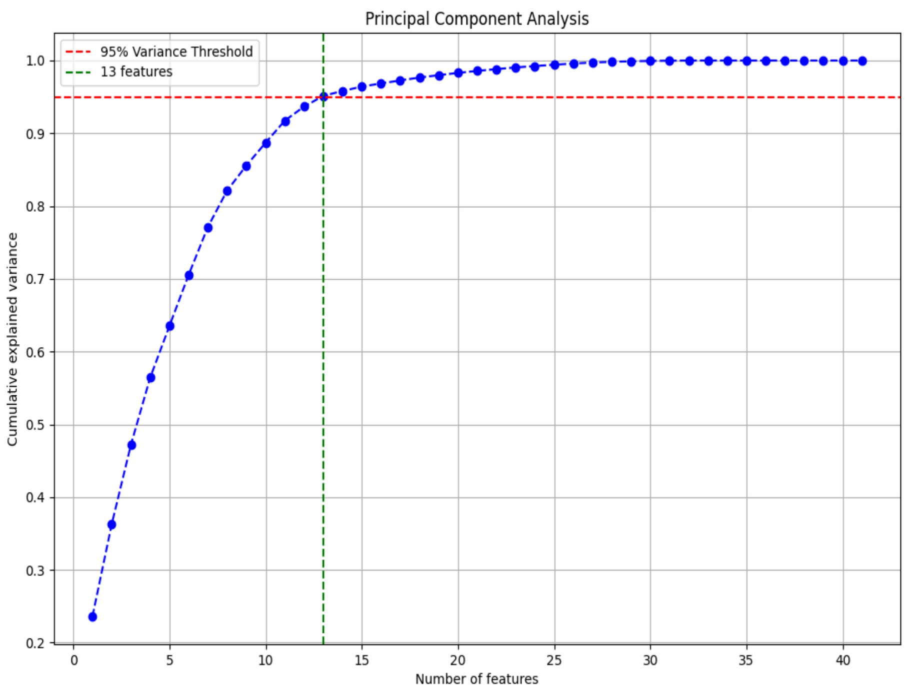
Eliminated such as ‘fastest_speed_down’, 'price', etc.

- The Random Forest analysis helped identify which features are most important for predicting upload speeds (**‘speed_up’**).
- Features like **‘speed_down’** and **‘fastest_speed_price’** played the biggest role in determining upload speeds.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of the dataset while retaining as much variance as possible.

PCA - dimension reduction plot -




```
************************ PRINCIPLE COMPONENT ANALYSIS *****************

Number of features needed to explain more than 95% of the variance: 13
```

- The results of applying PCA revealed that the selected components collectively account for at least **95% of the total variance** in the dataset.
- This means that the dimensionality of the data was effectively reduced while retaining most of the important information.
- After implementing PCA, the dataset was reduced to **13 features**, ensuring a more manageable and efficient dataset for further analysis and modeling.

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a technique used to simplify the dataset by reducing its dimensionality while keeping the most important information. After applying **SVD**, the dataset was reduced to the **top 12 components**.By selecting these 12 components, we retained the most important patterns in the data while reducing its dimensionality.

Features selected from SVD -

```
************************ SINGULAR VALUE DECOMPOSITION ************************

Selected Features:Index(['n_providers', 'ppl_per_sq_mile', 'block_group', 'lon',
       'internet_perc_broadband', 'race_perc_non_white', 'lat',
       'fastest_speed_down', 'fastest_speed_price', 'median_household_income',
       'speed_down', 'price'],
      dtype='object')
```

Variance Inflation Factor (VIF)

Variance Inflation Factor (VIF) is a technique used to assess multicollinearity in the dataset. High VIF values indicate that a feature is highly correlated with others, which can lead to redundancy and affect model performance.If a feature has a high VIF (usually above 5), it means it's strongly correlated with other features, which can cause redundancy.

Selected features using VIF -

```
************************ VARIANCE INFLATION FACTOR ************************

Variance Inflation Factor Table:
                  Variable       VIF
0                    const     0.000
1                      lat     1.329
2                      lon     1.468
3              block_group     1.195
4               speed_down     2.338
5                    price     1.001
6       fastest_speed_down 4,774.451
7      fastest_speed_price 4,811.750
8      race_perc_non_white     1.516
9          ppl_per_sq_mile     1.280
10             n_providers     1.197
11  internet_perc_broadband     1.478
12  median_household_income     1.019

Here the the vif scores for fastest_speed_down and fastest_speed_price are very high.
```

In my case, **'fastest_speed_down'** and **'fastest_speed_price'** had high VIF values, indicating they were highly correlated with other variables. As a result, I removed these features to reduce redundancy and make the dataset more effective for modeling.

Selection of the best feature selection method

After experimenting with various feature selection methods, including **Random Forest**, **PCA**, **SVD**, and **VIF**, I found that **VIF** provided the best results and performance metrics for my dataset in all the phases.

- **Phase II (Regression)**: VIF features were effective in regression tasks by identifying and removing multicollinear features..
- **Phase III (Classification)**: For classification, VIF helped improve the performance of all classifiers by eliminating redundant features.
- **Phase IV (Clustering)**: In clustering, VIF played a crucial role in ensuring that the features were independent, which enhanced the quality of the clusters formed.

Overall, I found features VIF, the most reliable and consistent feature selection method for my dataset, improving model accuracy while simplifying the data and reducing unnecessary complexity and redundancy.

Anomaly Detection and Removal

Anomaly detection is an essential step in data preprocessing to ensure the quality of the dataset and improve model accuracy. Outliers, or anomalies, can distort the results and affect the performance of the model.

In this project, **K-Means clustering** was used as a distance-based method to identify and remove anomalies, ensuring that the dataset used for further analysis is clean, reliable, and more representative of the underlying patterns.

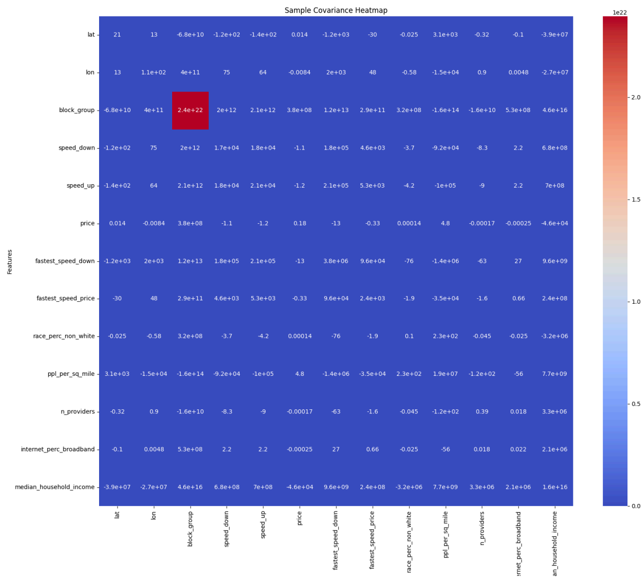
Removed all the anomalies/outliers -

```
************************ ANOMALY/OUTLIER ANALYSIS & REMOVAL ****
Number of anomalies detected: 17392
Shape of the dataset before anomaly removal: (347828, 19)
Shape of the dataset after anomaly removal: (330436, 19)
```

Sample Covariance Matrix

The **covariance matrix** helps to understand the relationships between numerical features in the dataset by showing how two features vary together.

The heatmap of covariance matrix of my dataset -

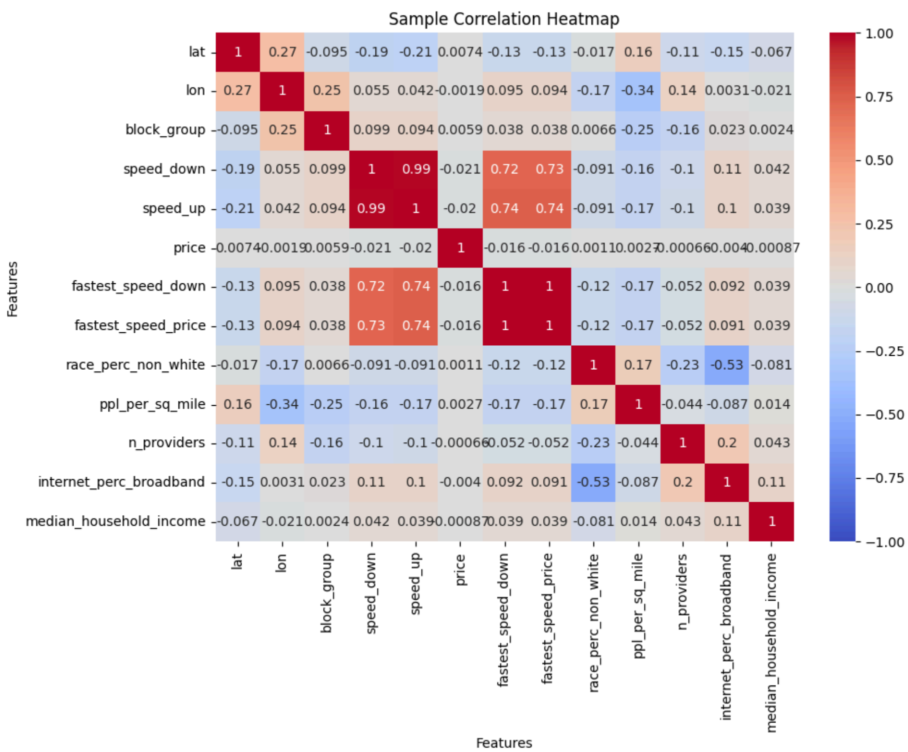


- Heatmap visualization shows the covariance values between the features, with most values close to zero, indicating weak correlations.
- The only significant covariance value is between block_group and itself, showing a perfect correlation. (highlighted red on the heatmap).

Sample Pearson's Correlation Coefficient Matrix

Pearson's correlation coefficient matrix is used to assess the linear relationships between numerical features in the dataset.

The heatmap of correlation coefficient matrix of the dataset -



- The correlation between features is mostly weak, with values closer to 0, indicating low linear relationships between most variables.

- The heatmap visualizes the coefficients ranging from -1 (negative correlation) to 1 (positive correlation).

- Some features show moderate correlations, while the rest have weak or no correlation.

Checking if the Target is Balanced

In **Phase III**, the target variable is **'package'**, which represents different types of internet plans and is categorical.
For classification tasks, it's important to verify if the target variable is balanced.

The value counts of **'package'** variable -

```
Internet Package class value counts for comparision:
package
Fiber Internet         147703
High-Speed Internet    104093
Basic Internet          78640
Name: count, dtype: int64
```

By analyzing the **value counts** of the different categories, we can ensure that no class is overrepresented or underrepresented.

PHASE II: REGRESSION ANALYSIS

In **Phase II**, I'll be focusing on **regression analysis** to predict **'speed_up'** (upload speeds) based on the features selected in the previous phase i.e., **Phase I**.
I'll be applying **multiple linear regression** to understand how other features affect upload speeds. The model's performance will be evaluated by performance metrics like **R-squared**, **Adjusted R-squared**, and **MSE.** Analysis of **T-test, F-test and Confidence Intervals** to understand the nature of the predictors on the target.
Backward stepwise regression will be used for eliminating irrelevant features and the final regression model will be fitted for more analysis.

Data Preparation

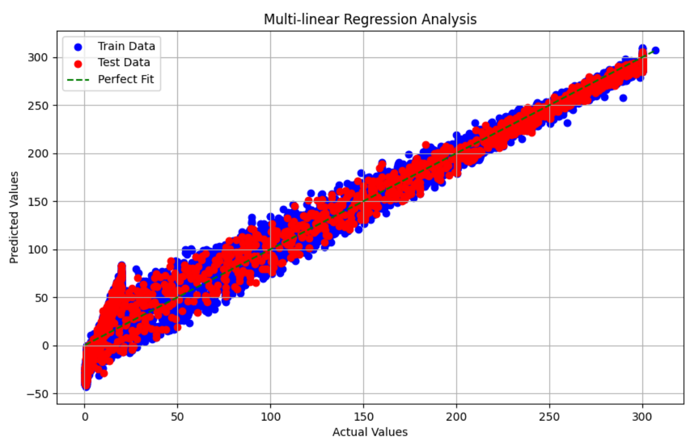
For **Phase II**, I used the same cleaning techniques from **Phase I** to prepare the data.

- **Feature selection -** based on VIF to eliminate multicollinearity and focus on the most important variables.
- **Data cleaning -** dropped unnecessary columns and handled the missing values by mode and median.
- **Data aggregation -** grouped by block_group with all the VIF selected features.
- **Outlier removal -** detected and removed all the outliers using K-means.
- **Target variable -** 'speed_up'
- **Split the dataset -** 80% training and 20% testing for the regression model.

Multiple Linear Regression

In **Phase II**, **Multiple Linear Regression** was applied to predict **'speed_up'** (upload speeds) using the features selected in the previous phases.
This technique helps to understand the relationship between the target variable and multiple predictors. By fitting the model to the data, we can evaluate how well the chosen features explain the variation in upload speeds.

Scatter Plot of train & test data -



This is the scatter plot which was created comparing the predicted values for the training and testing datasets which helps to visualize the model's performance better.

These are the performance metrics table -

```
Multiple Linear Regression Metrics table:
                         Value
Metric
R-squared                0.977
Adjusted R-squared       0.977
AIC                 74,558.373
BIC                 74,636.166
MSE                    300.839
```

● The model demonstrates a strong fit to the data, as shown by the high **R-squared** and **Adjusted R-squared** values.

● The **AIC** and **BIC** scores give us a sense of the model's complexity. While higher AIC and BIC values suggest the model is more complex.

● The **MSE (Mean Squared Error)** of around 300 indicates that the model has good predictive accuracy.

In conclusion, the regression model is well-tuned, with the right features contributing to accurate predictions of upload speeds.

T-test Analysis

The T-test evaluates the significance of each predictor variable in the regression model.

The t-values were calculated for each feature. Features with high values were eliminated, ensuring that the model only includes features with meaningful contributions.

The T-test values -

```
T-test results:
const                       0.733
block_group                -1.365
price                      -0.697
speed_down                571.482
n_providers                -6.147
ppl_per_sq_mile            -4.164
internet_perc_broadband   -12.170
lat                       -14.312
lon                        -5.076
race_perc_non_white       -11.132
median_household_income    -2.913
dtype: float64
```

According to the values, it is evident that **'speed_down'** has the most influence on the target variable **'speed_up'**.

F-test Analysis

The **F-test** evaluates the overall significance of the regression model whether the model as a whole provides a better fit to the data compared to a model with no predictors.

The F-test result -

```
F-test result:
36886.81452587509
```

The **F-test value** of **36,887** which is significantly high indicates that the model is statistically significant.

Confidence Interval Analysis

Confidence Interval Analysis helps identify whether the coefficients are statistically significant. If the interval for a feature's coefficient includes zero, it implies that the feature may not have a strong impact on predicting **'speed_up'** (upload speeds). This helps refine the model by focusing on features that provide more reliable and meaningful contributions to the target variable.

Confidence Intervals coefficients -

95% Confidence Intervals for Coefficients:

	0	1
const	-175.095	384.380
block_group	-0.000	0.000
price	-6.898	3.278
speed_down	1.085	1.093
n_providers	-0.022	-0.011
ppl_per_sq_mile	-0.000	-0.000
internet_perc_broadband	-20.474	-14.794
lat	-0.855	-0.649
lon	-0.153	-0.068
race_perc_non_white	-9.362	-6.558
median_household_income	-0.000	-0.000

It is evident that **'speed_down'** (download speeds) has a strong positive relationship with **'speed_up'** (upload speeds).

Stepwise Regression

Stepwise Regression is a technique used to improve a multiple linear regression model by selecting the most important features.I applied **Backward Elimination**, a method that starts with all predictors and removes the least significant ones based on their **p-values**. This process continues until only the most important variables remain. Eliminating features that didn't significantly contribute to predicting **'speed_up'.**

Features eliminated -

```
Eliminating price with a p-value of 0.4855
Eliminating block_group with a p-value of 0.1608
```

- In the **first iteration** of the **Backward Elimination** process, the feature **'price'** was eliminated due to its high **p-value** of 0.48, indicating it had minimal impact on predicting **'speed_up'**.
- Next, **'block_group'** was removed with a **p-value** of 0.16, as it also showed little statistical significance in contributing to the model.

After the removal of these features, the model was refitted, ensuring that only the most relevant predictors were included, improving the model's efficiency and performance.

The final model summary -

```
Final Model:
                            OLS Regression Results
==============================================================================
Dep. Variable:               speed_up   R-squared:                       0.977
Model:                            OLS   Adj. R-squared:                  0.977
Method:                 Least Squares   F-statistic:                 4.611e+04
Date:                Sat, 07 Dec 2024   Prob (F-statistic):               0.00
Time:                        21:25:00   Log-Likelihood:                -37269.
No. Observations:                8709   AIC:                         7.456e+04
Df Residuals:                    8700   BIC:                         7.462e+04
Df Model:                           8
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                       2.3766      3.301      0.720      0.472      -4.094       8.847
speed_down                  1.0893      0.002    572.871      0.000       1.086       1.093
n_providers                -0.0162      0.003     -6.049      0.000      -0.021      -0.011
ppl_per_sq_mile            -0.0002   3.81e-05     -4.022      0.000      -0.000   -7.86e-05
internet_perc_broadband   -17.7117      1.447    -12.237      0.000     -20.549     -14.874
lat                        -0.7233      0.048    -15.054      0.000      -0.817      -0.629
lon                        -0.1226      0.020     -6.236      0.000      -0.161      -0.084
race_perc_non_white        -7.9930      0.715    -11.186      0.000      -9.394      -6.592
median_household_income -3.852e-09   1.32e-09     -2.917      0.004   -6.44e-09   -1.26e-09
==============================================================================
Omnibus:                      198.988   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              212.493
Skew:                          -0.382   Prob(JB):                     7.21e-47
Kurtosis:                       3.030   Cond. No.                     2.63e+09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.63e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The final model shows an impressive **R-squared** and **Adjusted R-squared** of **97.7%**

By retaining only the most impactful predictors, **Stepwise Regression** ensured that the model maintains high accuracy and interpretability. This approach effectively eliminated irrelevant features.

PHASE III: CLASSIFICATION ANALYSIS

In **Phase III**, the goal is to predict the type of internet package, **'package'**, by applying classification models. The task is to classify the internet package classes – **Fiber Internet**, **High-Speed Internet**, and **Basic Internet**.

For this, I implemented seven different classifiers: **Pre-pruned Decision Tree**, **Post-pruned Decision Tree**, **Logistic Regression**, **K-Nearest Neighbors (KNN)**, **Support Vector Machine (SVM)**, **Naive Bayes**, and **Neural Networks**.

Each model is evaluated using key metrics like **accuracy**, **precision**, **recall**, **F1-score**, and **AUC**. The goal is to identify the best classifier for predicting the internet package.

Data Preparation

For **Phase III**, I followed the same cleaning techniques from **Phase I** to prepare the data for classification.

- **Feature selection -** based on VIF to eliminate multicollinearity and focus on the most important variables.
- **Data cleaning -** dropped unnecessary columns and handled the missing values by mode and median.
- **Data aggregation -** grouped by 'block_group' with all the VIF selected features.
- **Outlier removal -** detected and removed all the outliers using K-means.
- **Target variable -** 'package' – with three internet plans.
- **Split the dataset -** 80% training and 20% testing (with stratify=y) to maintain class distribution in the training and testing sets for classification models.

Pre-pruned Decision Tree Classifier

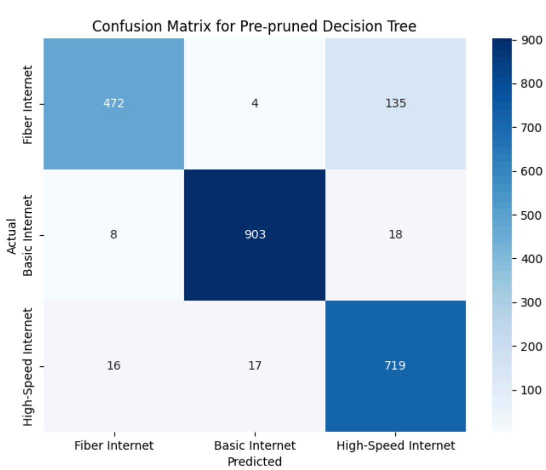
A **pre-pruned decision tree classifier** is a simple machine learning model that creates a decision tree but limits its size and complexity before it fully grows. This helps avoid overfitting and ensures the model generalizes better to new data.For this classifier, I used **grid search** to find the best hyperparameters and trained the model to predict the target.

Best parameters and the performance metrics for pre-pruned decision tree -

```
Starting Grid Search for Pre-pruned Decision Tree...

Best Parameters: {'criterion': 'gini', 'max_depth': 5,
 'max_features': 'sqrt', 'min_samples_leaf': 30,
 'min_samples_split': 20, 'splitter': 'best'}

Confusion Matrix:
[[472   4 135]
 [  8 903  18]
 [ 16  17 719]]
Train Accuracy: 0.9142
Test Accuracy: 0.9136
Precision: 0.9203
Recall: 0.9136
Specificity: 0.9575
F1-score: 0.9129
AUC: 0.981674001126671
```

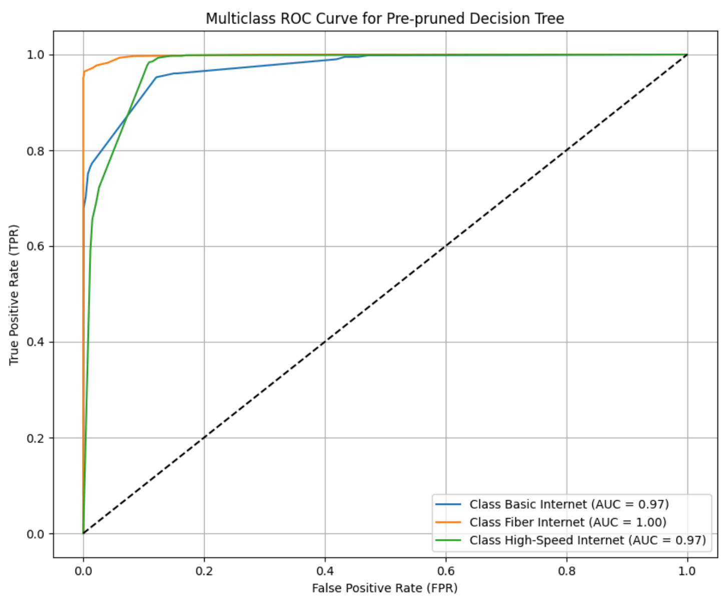
- According to the metrics, the model shows strong performance based on the high **AUC** score of 0.9817.
- Both **train and test accuracy** are high, indicating the model generalizes well.
- The **precision** and **recall** scores reflect the model's ability to correctly identify the classes, especially **Fiber Internet**, which shows excellent prediction results.

Confusion matrix heatmap for pre-pruned decision tree -



- Confusion Matrix Heatmap represents the model's performance in predicting the target classes.
- The classifier performances exceptionally well for the Fiber Internet class with 903 correct predictions.
- There is some misclassification in predicting Basic Internet and High-Speed Internet, as observed in the off-diagonal values.

The ROC Curve Plot -



The **AUC** scores for each class are:

- **Basic Internet**: 0.97
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 0.97

- The Fiber Internet class achieves a perfect AUC score of 1.00 indicating flawless discrimination by the model.
- Both Basic Internet and High-Speed Internet also demonstrate strong performance with AUC scores of 0.97.

This **model** performs strongly for **Fiber Internet**. Some misclassifications between **Basic Internet** and **High-Speed Internet** were observed, but the overall model performance is reliable, with high **precision**, **recall**, and **AUC**.

Post-pruned Decision Tree classifier

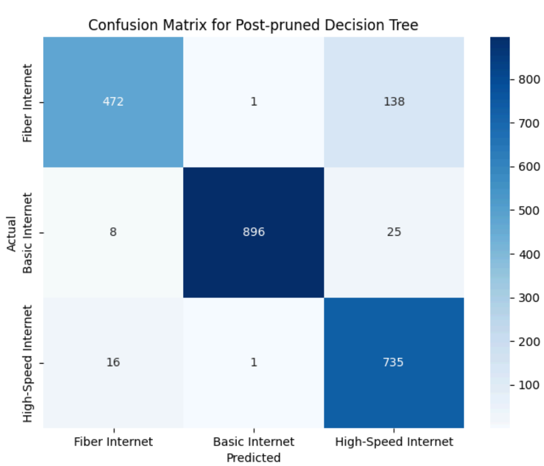
I used the **Post-pruned Decision Tree Classifier** to predict the target variable **'package'**. To improve the model, I tuned its hyperparameters using **grid search**, focusing on selecting the optimal **ccp_alpha** (cost-complexity pruning alpha) value. This value helps prune the tree, ensuring it doesn't become too complex and overfit the data.

Best parameters and the performance metrics for post-pruned decision tree -

```
Starting Grid Search for Post-pruned Decision Tree...

Best Parameters: {'criterion': 'gini', 'max_depth': 5,
 'max_features': 'sqrt', 'min_samples_leaf': 30,
 'min_samples_split': 20, 'splitter': 'best'}

Confusion Matrix:
[[472   1 138]
 [  8 896  25]
 [ 16   1 735]]
Train Accuracy: 0.9138
Test Accuracy: 0.9175
Precision: 0.9266
Recall: 0.9175
Specificity: 0.9591
F1-score: 0.9172
AUC: 0.9804987714755086
```

- The **model** demonstrates strong performance with a high **AUC** of 0.9805.
- Both **train and test accuracy** are consistent, which suggests that the model generalizes well to unseen data.
- The **precision** and **recall** values, and **specificity** shows that the model is great.

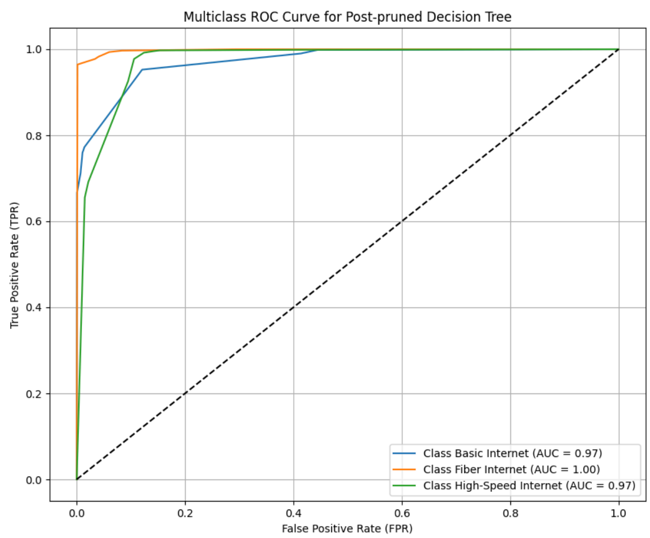
Confusion matrix heatmap for pre-pruned decision tree -



- Confusion matrix shows the classifier's prediction for each class.

- Fiber internet class shows excellent performance with 896 correct predictions.
- Basic Internet has some misclassifications, but still performs reasonably well.
- The model classifies High-Speed internet with high accuracy.

The ROC Curve Plot -



The **ROC curve** shows the **AUC** scores for each class:

- **Basic Internet**: 0.97
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 0.97

- The Fiber Internet has a perfect AUC score of 1.00 indicating flawless classification performance.
- Both Basic Internet and High-Speed Internet show strong AUC scores of 0.97.

The **model** shows great performance, particularly for **Fiber Internet**, where it achieves near-perfect accuracy. While there are some misclassifications between **Basic Internet** and **High-Speed Internet**, the overall results are solid, with high **precision**, **recall**, and **AUC** scores. The post-pruning technique helped optimize the model, reducing complexity.

Logistic Regression

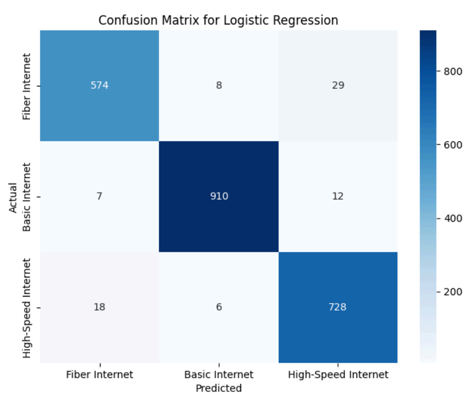
Logistic Regression is a statistical model commonly used for binary and multi-class classification tasks, which estimates the probability of a target variable of a particular class. To improve the model, I applied **grid search** to find the best hyperparameters, focusing on optimizing the **C** parameter and using the **l2 penalty** for regularization.

Best parameters and the performance metrics for Logistic regression -

```
Starting Grid Search for Logistic Regression...

Best Parameters: {'C': 10, 'penalty': 'l2'}

Confusion Matrix:
[[574   8  29]
 [  7 910  12]
 [ 18   6 728]]
Train Accuracy: 0.9596
Test Accuracy: 0.9651
Precision: 0.9652
Recall: 0.9651
Specificity: 0.9825
F1-score: 0.9651
AUC: 0.9967063042757225
```

- The test accuracy of 96.51% indicates the model's excellent generalization to new data.
- The AUC score of 0.9967 shows exceptional performance.

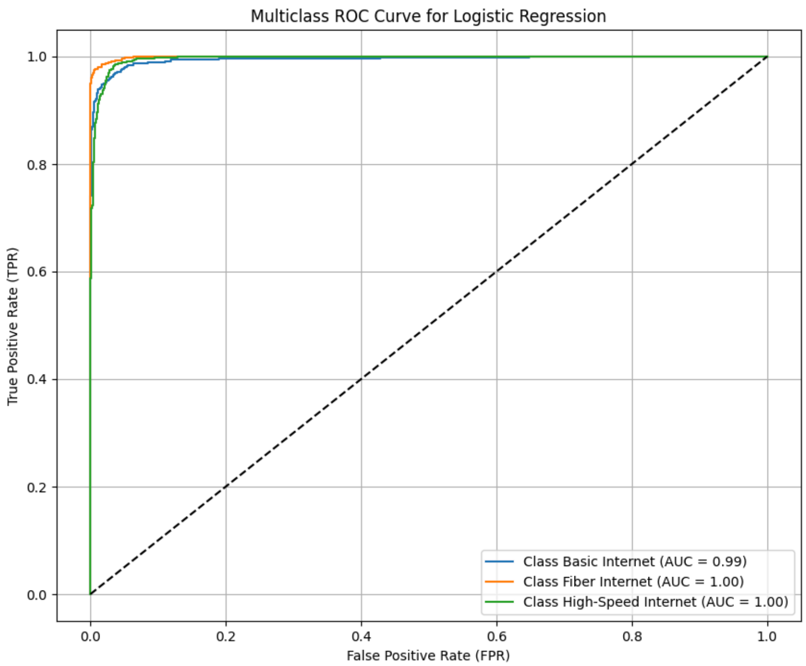
Confusion matrix heatmap for Logistic Regression -



- Fiber internet and High-Speed Internet classes have high accuracy, with 910 and 728 correct predictions, respectively..

- Basic Internet shows some misclassifications, but overall performance is strong.

The ROC Curve Plot -



The ROC curve shows the AUC scores for each class:

- **Basic Internet**: 0.99
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 1.00

- Fiber Internet and High-Speed Internet have perfect AUC scores of 1.00 indicating flawless classification performance.

- Both Basic Internet shows excellent performance with an AUC score of 0.99.

The **Logistic Regression** model performs excellently, with high **accuracy**, **precision**, and **recall** across all classes. The **ROC curve** further confirms its ability to differentiate between **internet plan types**. The fine-tuning of the hyperparameters helped optimize the model, making it a reliable choice for classifying **'package'**.

K-Nearest Neighbors (KNN)

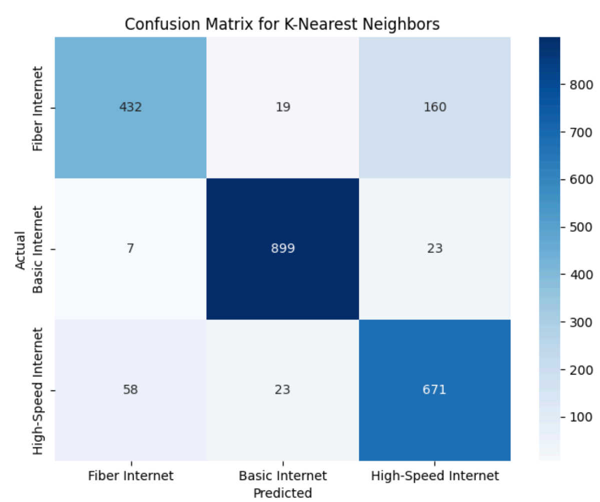
K-Nearest Neighbors (KNN) is a simple yet powerful classification algorithm that assigns a class to a data point based on the majority class of its nearest neighbors. To improve the model, I used **grid search** to find the optimal hyperparameters, focusing on the number of neighbors (**n_neighbors**) and the distance-based weighting.

Best parameters and the performance metrics for KNN -

```
Starting Grid Search for K-Nearest Neighbors...

Best Parameters: {'algorithm': 'auto', 'n_neighbors': 11,
 'weights': 'distance'}

Confusion Matrix:
[[432  19 160]
 [  7 899  23]
 [ 58  23 671]]
Train Accuracy: 1.0000
Test Accuracy: 0.8735
Precision: 0.8767
Recall: 0.8735
Specificity: 0.9371
F1-score: 0.8718
AUC: 0.9676117729536106
```

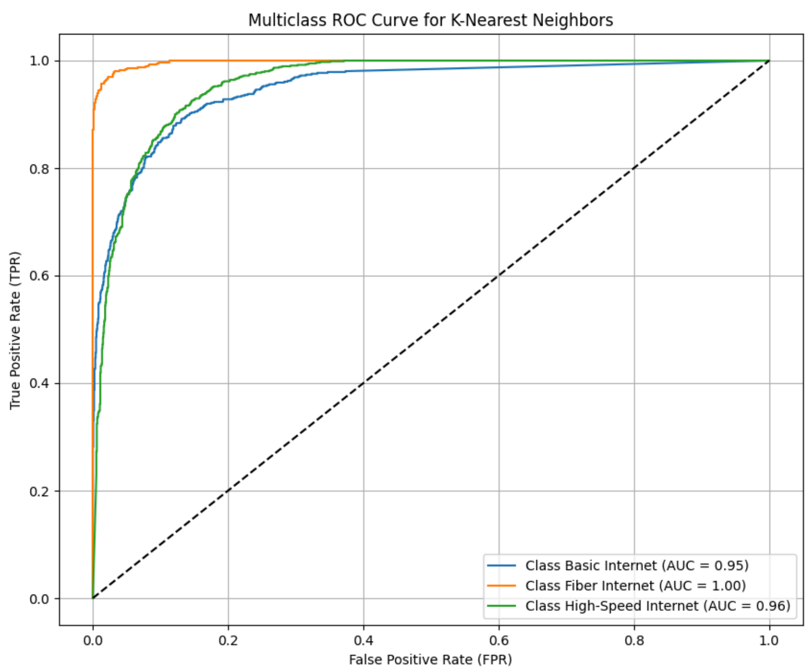
- The model achieves perfect accuracy on training data (1.00), but shows a test accuracy of 87.35% indicating some overfitting.
- AUC score of 0.9676 indicates strong performance in classification.

Confusion matrix heatmap for KNN -



- The model performs well for Fiber Internet and High-Speed Internet classes, with high correct predictions.
- Basic internet shows some misclassification, but the overall model is strong.

The ROC Curve Plot -



The ROC curve shows the AUC scores for each class:

- **Basic Internet**: 0.95
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 0.96

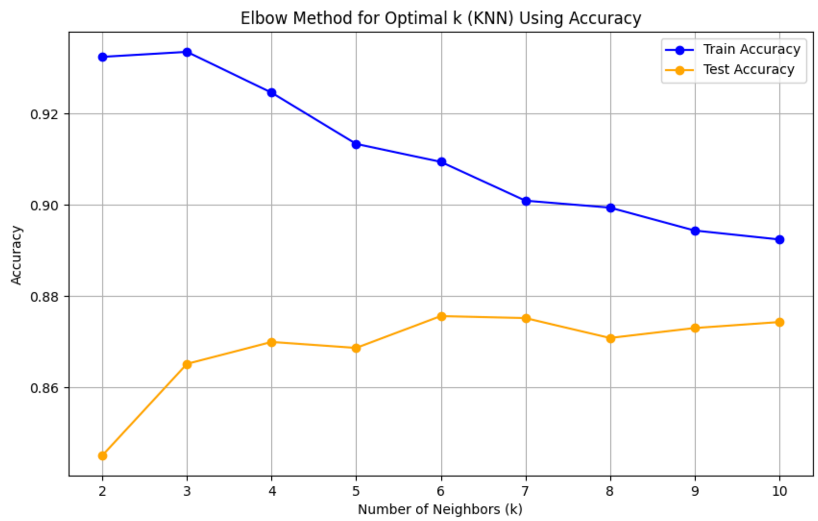
- Fiber Internet has a perfect AUC score of 1.00.
- Basic Internet and High-Speed Internet have strong AUC scores (0.95, 0.96), indicating good classification performance.

The **KNN** model works well with high performance for **Fiber Internet** and decent performance for **Basic Internet** and **High-Speed Internet**. However, the perfect **train accuracy** indicates overfitting, and the slight drop in **test accuracy** suggests that **KNN** may need tuning.

Optimum K for KNN (using elbow method)

The **Elbow Method** is a technique used to determine the optimal number of clusters (or neighbors in KNN) by plotting the **Within-Cluster Sum of Squares (WCSS)** against the number of neighbors (**k**). As **k** increases, the WCSS decreases, but the rate of decrease slows down after a certain point. The "elbow" point on the graph indicates the optimal **k**, where adding more neighbors doesn't significantly improve the model's performance.

The elbow plot for KNN -




```
Optimal k for KNN (using Elbow Method with Accuracy): 6
```

For my **KNN** model, the **optimal k** was determined to be **6**, as this provided the best balance between model complexity and classification accuracy.

Support Vector Machine (SVM)

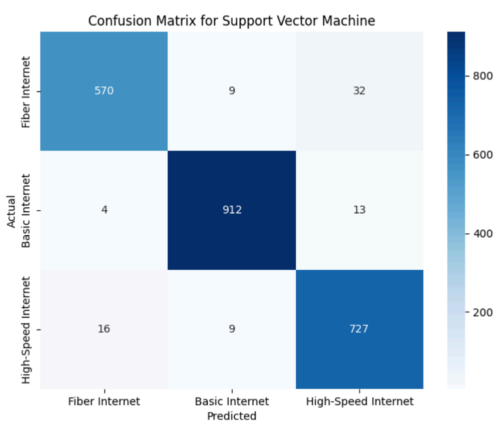
Support Vector Machine (SVM) is a powerful classification algorithm that works by finding the hyperplane that best separates data points of different classes. To improve the model, I applied **grid search** on kernels such as **linear, radial basis function (RBF),** and **polynomial,** to find the best hyperparameters (kernel), as it determines the decision boundary. In this case, after grid search, the **linear kernel** is the best kernel as it works well with my dataset.

Best parameters and the performance metrics for SVM -

```
Starting Grid Search for Support Vector Machine...

Best Parameters: {'kernel': 'linear'}

Confusion Matrix:
[[570   9  32]
 [  4 912  13]
 [ 16   9 727]]
Train Accuracy: 0.9568
Test Accuracy: 0.9638
Precision: 0.9640
Recall: 0.9638
Specificity: 0.9821
F1-score: 0.9638
AUC: 0.9968979470532524
```

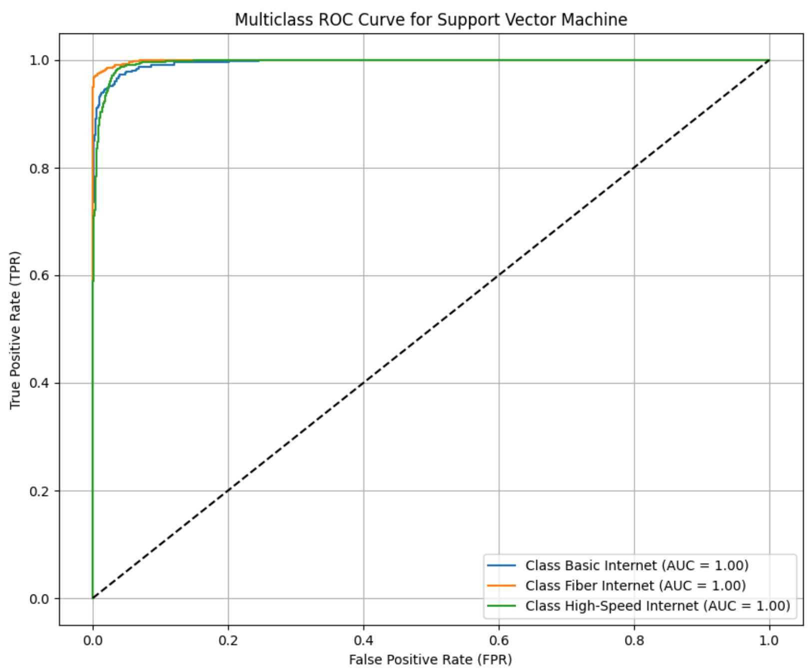
- According to the best parameters, the linear kernel was chosen for its simplicity and effectiveness in separating the classes in a hyperplane.
- Test accuracy of 96.38% indicates excellent model performance.

Confusion matrix heatmap for SVM -



- Fiber Internet has excellent accuracy with 912 correct predictions.
- Basic Internet shows some misclassification, but the performance is still very strong.
- High-Speed Internet also shows a high level of correct predictions.

The ROC Curve Plot -



The ROC curve shows the AUC scores for each class:

- **Basic Internet**: 1.00
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 1.00

All three classes show perfect **AUC** scores of 1.00, indicating that the SVM classifier can flawlessly classify between internet packages.

The **SVM** model performed excellently, with high **AUC**, **precision**, and **recall** scores across all classes. The **linear kernel** was particularly effective, providing perfect classification for **Fiber Internet** and **High-Speed Internet**. This makes the **SVM** a strong choice for classifying internet plans in this dataset.

Naive Bayes Classifier

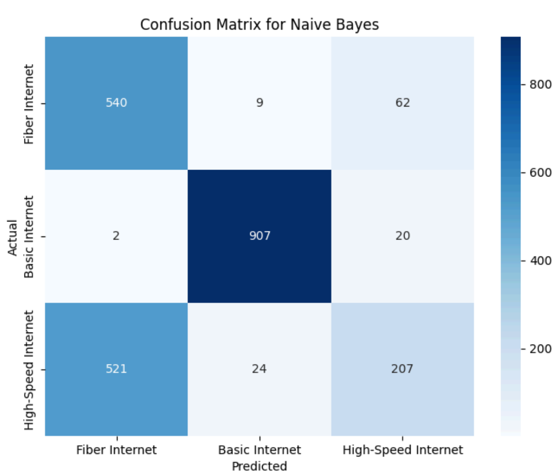
Naive Bayes is a probabilistic classification model based on applying Bayes' theorem with strong (naive) independence assumptions between the features. I applied **grid search** to tune the hyperparameters, specifically focusing on the **var_smoothing** parameter, which helps prevent division by zero errors and smoothens the likelihood estimates.

Best parameters and the performance metrics for Naive Bayes -

```
Starting Grid Search for Naive Bayes...

Best Parameters: {'var_smoothing': 1e-07}

Confusion Matrix:
[[540   9  62]
 [  2 907  20]
 [521  24 207]]
Train Accuracy: 0.7232
Test Accuracy: 0.7216
Precision: 0.7615
Recall: 0.7216
Specificity: 0.8500
F1-score: 0.6959
AUC: 0.9392826456562188
```

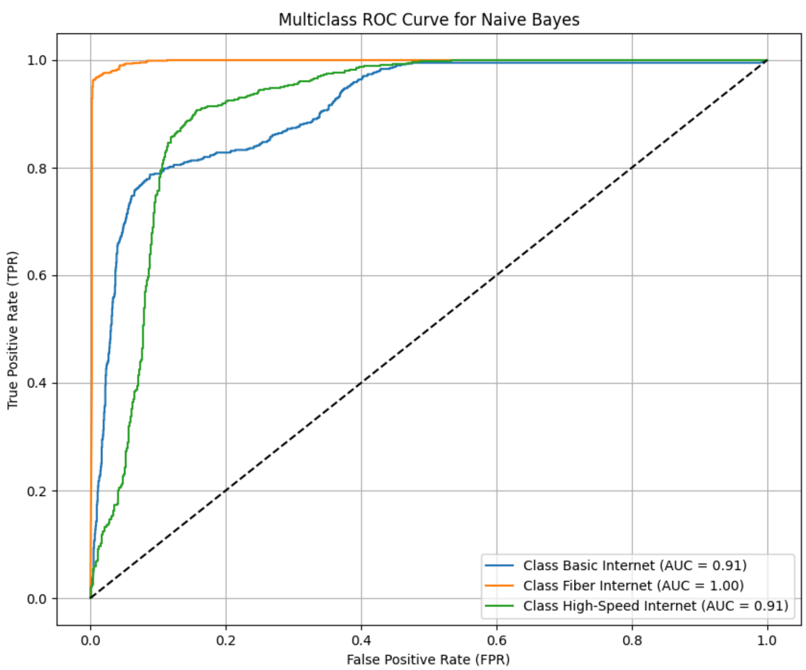
- The model shows relatively low train and test accuracy that is around 72%, indicating potential underfitting.
- However, the AUC score of 0.9393 demonstrates strong classifying power between classes.

Confusion matrix heatmap for Naive Bayes -



- Fiber internet class performs well with 907 correct predictions.
- Basic Internet shows considerable misclassification with 521 classified as High-Speed Internet.
- High-Speed Internet also has some misclassification but with overall reasonable accuracy.

The ROC Curve Plot -



The ROC curve shows the AUC scores for each class:

- **Basic Internet**: 0.91
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 0.91

- Fiber Internet has a perfect AUC score of 1.00, indicating flawless classification.
- Basic Internet and High-Speed Internet both have AUC scores of 0.91, showing strong but less perfect performance.

The **Naive Bayes** model provides decent performance, with perfect classification for **Fiber Internet** and slightly lower accuracy for **Basic Internet** and **High-Speed Internet**.

Neural Networks Classifier

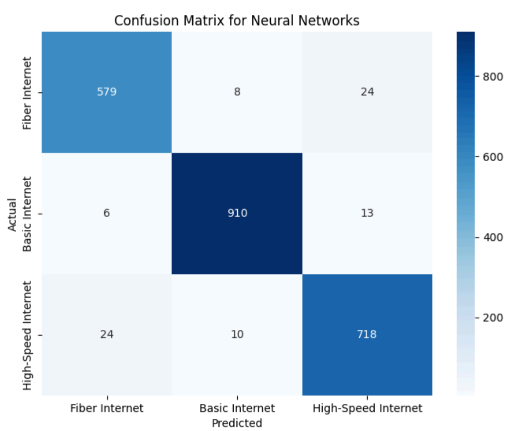
Neural Networks are a powerful family of machine learning models inspired by the human brain. They consist of layers of interconnected nodes (neurons), where each connection has a weight that is adjusted during training. To optimize the model, I applied **grid search** to find the best hyperparameters, particularly focusing on the **number of hidden layers**, **activation function**, and **learning rate**.

Best parameters and the performance metrics for Neural Networks -

```
Best Parameters: {'activation': 'relu', 'alpha': 0.0001,
 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

Confusion Matrix:
[[579   8  24]
 [  6 910  13]
 [ 24  10 718]]
Train Accuracy: 0.9691
Test Accuracy: 0.9629
Precision: 0.9629
Recall: 0.9629
Specificity: 0.9814
F1-score: 0.9629
AUC: 0.9967151746982684
```

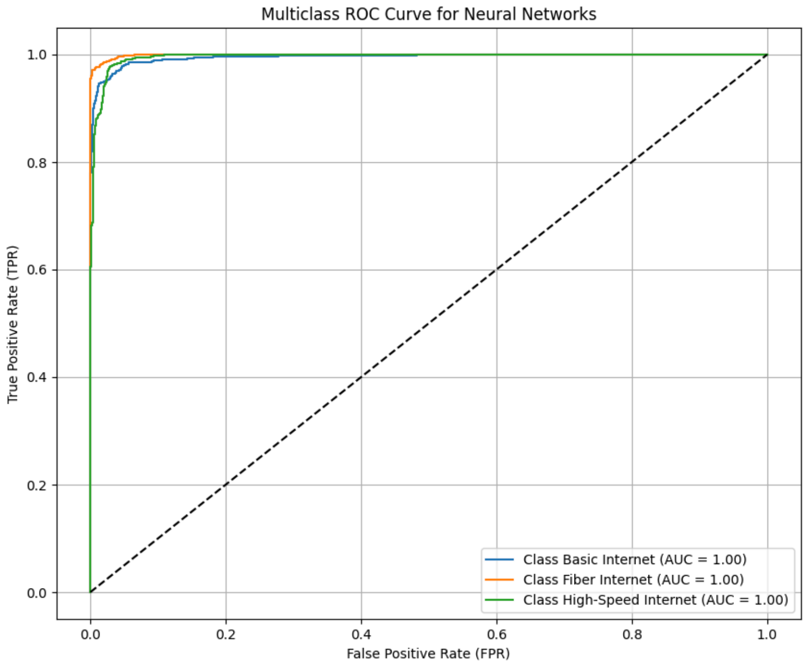
- Test accuracy of 96.29% shows excellent model generalization.
- The AUC score of 0.9967 indicates outstanding performance, with perfect AUC for all classes (1.00).

Confusion matrix heatmap for Neural Networks -



- Fiber Internet class performance excellently with 910 correct predictions.
- Basic Internet and High-Speed Internet also show strong performance, with low misclassification.

The ROC Curve Plot -



The ROC curve shows the AUC scores for each class:

- **Basic Internet**: 1.00
- **Fiber Internet**: 1.00
- **High-Speed Internet**: 1.00

The model shows perfect classification for all classes with AUC = 1.00 across Basic Internet, Fiber Internet, and High-Speed Internet, indicating flawless classification by the neural network.

The **Neural Networks** classifier performed exceptionally well, with **high accuracy** and **AUC scores** of 1.00 for all classes.

Comparison of Models and Selection of the Best Classifier

In this section, I will compare the performance of all the classifiers used in this project: **Pre-pruned Decision Tree**, **Post-pruned Decision Tree**, **Logistic Regression**, **K-Nearest Neighbors (KNN)**, **Support Vector Machine (SVM)**, **Naive Bayes**, and **Neural Networks**. The goal is to evaluate which classifier performed the best across various metrics such as **accuracy**, **precision**, **recall**, **AUC**, and **F1-score**.

The performance metrics table for all the classifiers -

Model	Train Accuracy	Test Accuracy	Precision	Recall
Pre-pruned Decision Tree	0.914	0.914	0.920	0.914
Post-pruned Decision Tree	0.914	0.918	0.927	0.918
Logistic Regression	0.960	0.965	0.965	0.965
K-Nearest Neighbors	1.000	0.873	0.877	0.873
Support Vector Machine	0.957	0.964	0.964	0.964
Naive Bayes	0.723	0.722	0.762	0.722
Neural Networks	0.969	0.963	0.963	0.963

Model	Specificity (Weighted)	F1-score	AUC
Pre-pruned Decision Tree	0.958	0.913	0.982
Post-pruned Decision Tree	0.959	0.917	0.980
Logistic Regression	0.982	0.965	0.997
K-Nearest Neighbors	0.937	0.872	0.968
Support Vector Machine	0.982	0.964	0.997
Naive Bayes	0.850	0.696	0.939
Neural Networks	0.981	0.963	0.997

Pre-pruned DT: Has good accuracy and strong AUC, but prone to overfit, especially for the Basic Internet class.

Post-pruned DT: The test accuracy improved but prone to underfit if pruning is too aggressive.

Logistic Regression: High test accuracy and AUC with strong generalization.

KNN: Perfect train accuracy, but lower test accuracy, indicating overfitting.

SVM: Strong test accuracy and AUC across all classes, but computationally expensive.

Naives Bayes: Struggled with lower accuracy and had lower AUC for certain classes.

Neural Networks: Showed perfect AUC for all classes and high test accuracy, but computationally expensive.

The **best classifier** in my opinion is **Logistic Regression** due to its strong performance with high test accuracy and AUC across all classes. It has the ability to generalize well to new data.

PHASE IV: CLUSTERING & ASSOCIATION

In **Phase IV**, the focus is on analyzing the dataset through unsupervised learning techniques like **Clustering** and **Association Rule Mining**. The goal is to understand underlying patterns and relationships within the data by implementing these algorithms and models.

For clustering, I am going to apply: **K-Means** and **DBSCAN**, to group similar observations based on their features.

Additionally, I am implementing the **Apriori algorithm** for association rule mining, focusing on categorical features like **package**, **technology**, and **speed_category**, to discover meaningful relationships like market-basket analysis between these variables.

Data Preparation

- **Feature selection –** Dropped categorical features and irrelevant numeric features for clustering based on VIF.
- **Data cleaning –** Dropped unnecessary columns and handled missing values by filling with median or mean.
- **Feature engineering –** Created a new feature, 'speed_category', by classifying 'speed_down' as 'Slow' or 'Fast'.
- **Downsampling –** Reduced the dataset size to 30% for faster computation in clustering.
- **Clustering dataset preparation –** Created a copy of the downsampled data for clustering and standardized the features.
- **Target variable –** No target variable used for clustering, focusing on unsupervised learning.
- **Standardization –** Standardized the features using StandardScaler for fair contribution to clustering.

K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm used to group data into clusters based on similarity.

To find the Optimal number of clusters (K), there are two methods -

- Elbow method (WCSS)
- Silhouette Score method

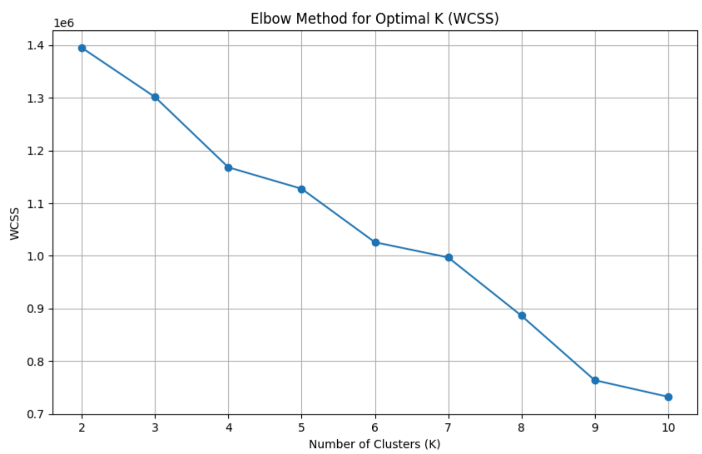
Elbow Method

To determine the ideal number of clusters, we analyze how the Within-Cluster Sum of Squares (WCSS) changes with increasing K.

As K increases, WCSS decreases, but it slows down after a certain point.

This point is known as the elbow, which represents the optimal K.

The elbow graph -




```
Optimal K (using WCSS - Elbow Method): 6
```

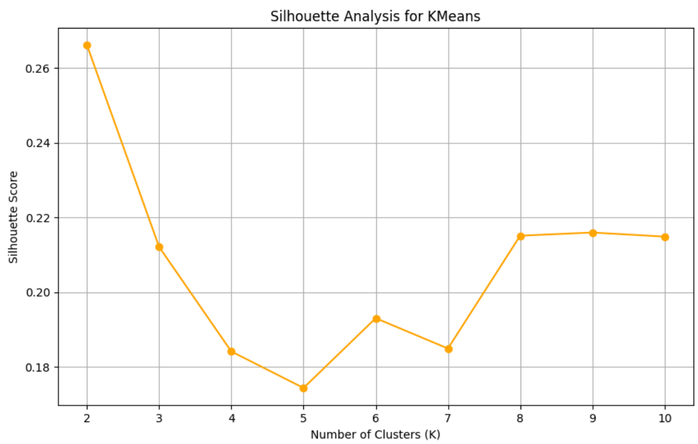
The elbow graph shows a distinct elbow at K=6 suggesting it as the optimal number of clusters.

Silhouette Score Method
To determine the number of clusters, we measure the silhouette scores.

Higher scores indicate better-defined clusters, with a value close to 1.0 being ideal.

By analyzing scores for K 2 to 10, we identify K that yields the highest Silhouette Score as the optimal number of clusters.

Silhouette Score graph -




```
Optimal K (using Silhouette Score): 2
```

The Silhouette graph indicates the highest score at K=2 suggesting it as the optimal number of clusters.

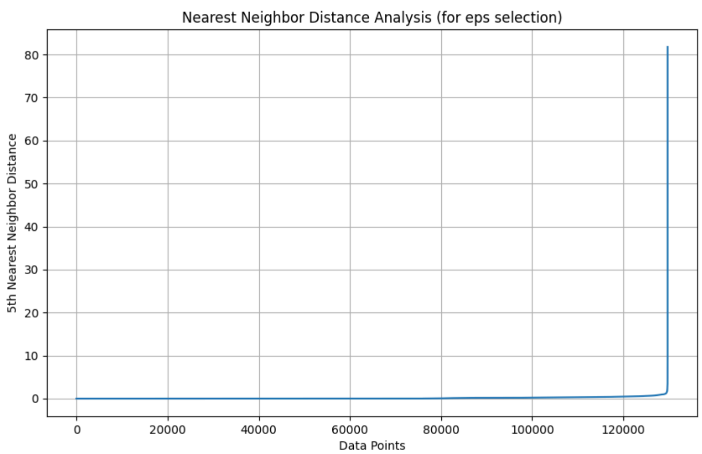
DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised learning algorithm that groups data points into clusters based on density.
It relies on two parameters, **eps** (epsilon) and **min_samples** (the minimum number of points required to form a dense region).

Parameter Selection:

- I set **eps = 1.5** and **min_samples = 8** based on a nearest-neighbor (**5**) distance analysis plot.
- These parameters ensure a balance between capturing dense regions and isolating noise points.

Nearest Neighbor Distance Graph for epsilon selection -




```
DBSCAN Number of clusters: 95
DBSCAN Number of noise points: 398
```

It identified 95 clusters in the dataset and 398 noise points detected.

Association Rule Mining

Association Rule Mining is an unsupervised learning technique used to discover relationships and co-occurrences between items in a dataset. It identifies frequent itemsets and generates rules that explain the likelihood of items appearing together. This method is particularly useful for analyzing categorical data.
I used Apriori Algorithm to mine rules from three categorical features- **'package'**, **'technology'**, and **'speed_category'**.

Apriori Algorithm

- Applied the Apriori algorithm with **min_support = 0.3** to generate frequent itemsets.
- Extracted association rules with a **confidence threshold of 0.7**.

These are the frequent itemsets -

```
*********************** APRIORI ALGORITHM ***********************

Frequent Itemsets:
    support                                          itemsets
0  0.352970                           (package_Fiber Internet)
1  0.353602                                 (technology_Fiber)
2  0.453725                             (technology_Not Fiber)
3  0.514754                               (speed_category_Slow)
4  0.485246                               (speed_category_Fast)
5  0.352954         (technology_Fiber, package_Fiber Internet)
6  0.352970      (package_Fiber Internet, speed_category_Fast)
7  0.353147            (technology_Fiber, speed_category_Fast)
8  0.321626       (speed_category_Slow, technology_Not Fiber)
9  0.352954  (technology_Fiber, package_Fiber Internet, spe...
```

- **package_Fiber Internet** and **technology_Fiber** have a **support of 35.30%**, highlighting their frequent co-occurrence in the dataset.
- **speed_category_Fast** and **technology_Fiber** also appear together frequently with a support of **35.31%**, indicating a preference for fast speeds among Fiber Internet users.
- **speed_category_Slow** and **technology_Not Fiber** have a support of **32.16%**, reflecting associations between slower speeds and non-Fiber technology.

These are the association rules -

```
Association Rules:
                                  antecedents  \
0                          (technology_Fiber)
1                    (package_Fiber Internet)
2                    (package_Fiber Internet)
3                       (speed_category_Fast)
4                          (technology_Fiber)
5                       (speed_category_Fast)
6                      (technology_Not Fiber)
7     (package_Fiber Internet, technology_Fiber)
8        (technology_Fiber, speed_category_Fast)
9  (package_Fiber Internet, speed_category_Fast)
10                         (technology_Fiber)
11                   (package_Fiber Internet)
12                      (speed_category_Fast)

                                  consequents   support  confidence  \
0                    (package_Fiber Internet)  0.352954    0.998168
1                          (technology_Fiber)  0.352954    0.999956
2                       (speed_category_Fast)  0.352970    1.000000
3                    (package_Fiber Internet)  0.352970    0.727404
4                       (speed_category_Fast)  0.353147    0.998713
5                          (technology_Fiber)  0.353147    0.727770
6                       (speed_category_Slow)  0.321626    0.708857
7                       (speed_category_Fast)  0.352954    1.000000
8                    (package_Fiber Internet)  0.352954    0.999454
9                          (technology_Fiber)  0.352954    0.999956
10 (package_Fiber Internet, speed_category_Fast)  0.352954    0.998168
11       (technology_Fiber, speed_category_Fast)  0.352954    0.999956
12    (package_Fiber Internet, technology_Fiber)  0.352954    0.727372

        lift
0   2.827915
1   2.827915
2   2.060812
3   2.060812
4   2.058160
5   2.058160
6   1.377079
7   2.060812
8   2.831558
9   2.827915
10  2.827915
11  2.831558
12  2.060812
```

- Fiber technology is highly predictive of Fiber Internet packages.
- Fast speed categories are frequently linked with Fiber technology and premium plans like Fiber Internet.

RECOMMENDATIONS

a. What did you learn from this project?

- I gained hands-on experience in data cleaning, data preprocessing, data engineering, and data manipulation.Additionally, I implemented various methods for feature selection in the first phase which augmented my knowledge.

In phase II, I learned about the regression analysis, more detailed study on performance metrics and analysis about it enhancing my knowledge for analyzing effectively.

In phase III, I explored the characteristics of different classifiers and the concept of grid search for hyperparameter optimization. I learned how classifiers work and how their performance can be fine-tuned.

In phase IV, I learned about the uncovering associations within the dataset in the feature matrix, and applied clustering algorithms and analyze them effectively.

b. Which classifiers perform the best for the selected dataset?

- SVM, Neural Networks and Logistic all perform the best for my dataset. SVM and Neural Networks are computationally expensive. I would choose Logistic Regression as it has the generalization ability and very good accuracy.

c. How do you think you can improve the performance of the classification? This could be in the future work section.

- Trial and error is the only key, by passing in different parameters, for grid search until we get the best outcome for the desired dataset.

d. What features are associated with the target variable?

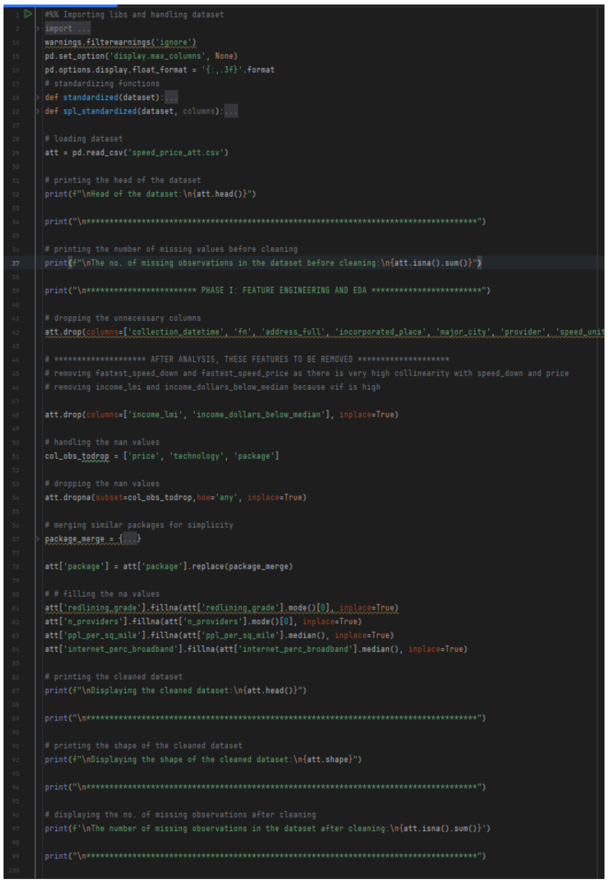
- Technology and Speed Category are associated with the target variable in association rule mining. For instance, Fast speed categories are linked with Fiber Internet packages.

e. Number of clusters in this feature space.

- Number of clusters shown in the DBSCAN are 95.

APPENDIX

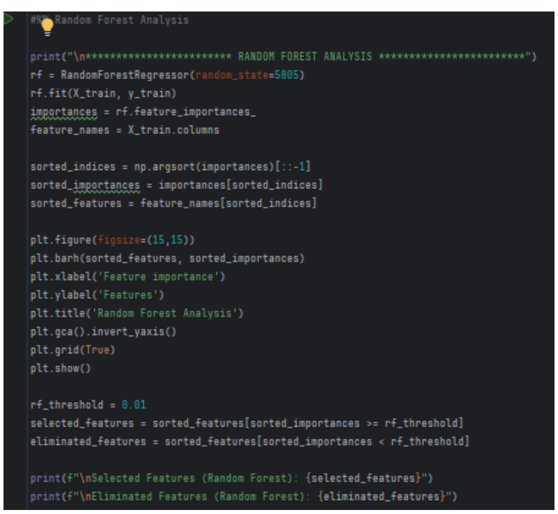
PHASE I - Feature Engineering - Importing, cleaning, handling dataset and splitting



```python
#%% Importing libs and handling dataset
import ...
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.options.display.float_format = '{:,.3f}'.format
# standardizing functions
def standardized(dataset):...
def spl_standardized(dataset, columns):...

# loading dataset
att = pd.read_csv('speed_price_att.csv')

# printing the head of the dataset
print(f"\nHead of the dataset:\n{att.head()}")

print("\n************************************************************************************")

# printing the number of missing values before cleaning
print(f"\nThe no. of missing observations in the dataset before cleaning:\n{att.isna().sum()}")

print("\n************************* PHASE I: FEATURE ENGINEERING AND EDA *************************")

# dropping the unnecessary columns
att.drop(columns=['collection_datetime', 'fn', 'address_full', 'incorporated_place', 'major_city', 'provider', 'speed_unit

# ******************** AFTER ANALYSIS, THESE FEATURES TO BE REMOVED ********************
# removing fastest_speed_down and fastest_speed_price as there is very high collinearity with speed_down and price
# removing income_lmi and income_dollars_below_median because vif is high

att.drop(columns=['income_lmi', 'income_dollars_below_median'], inplace=True)

# handling the nan values
col_obs_todrop = ['price', 'technology', 'package']

# dropping the nan values
att.dropna(subset=col_obs_todrop,how='any', inplace=True)

# merging similar packages for simplicity
package_merge = {...}

att['package'] = att['package'].replace(package_merge)

# # filling the na values
att['redlining_grade'].fillna(att['redlining_grade'].mode()[0], inplace=True)
att['n_providers'].fillna(att['n_providers'].mode()[0], inplace=True)
att['ppl_per_sq_mile'].fillna(att['ppl_per_sq_mile'].median(), inplace=True)
att['internet_perc_broadband'].fillna(att['internet_perc_broadband'].median(), inplace=True)

# printing the cleaned dataset
print(f"\nDisplaying the cleaned dataset:\n{att.head()}")

print("\n************************************************************************************")

# printing the shape of the cleaned dataset
print(f"\nDisplaying the shape of the cleaned dataset:\n{att.shape}")

print("\n************************************************************************************")

# displaying the no. of missing observations after cleaning
print(f'\nThe number of missing observations in the dataset after cleaning:\n{att.isna().sum()}')

print("\n************************************************************************************")
```

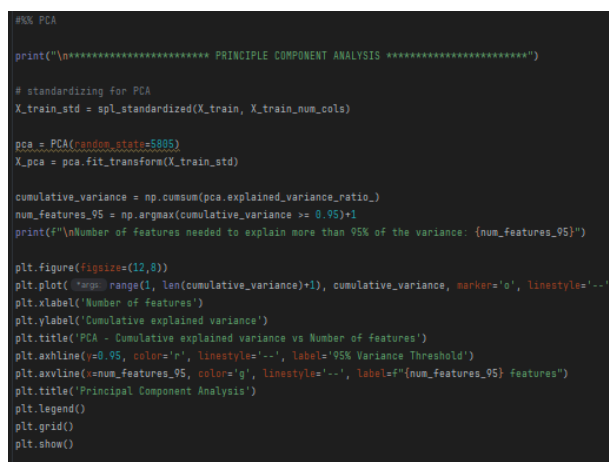
```python
# checking whether the dataset has duplicate observations
print(f"\nChecking whether the dataset has any duplications (before): {att.duplicated().sum()}")

att.drop_duplicates(inplace=True)

print("\n************************************************************************************")

print(f"\nChecking whether the dataset has any duplications (after): {att.duplicated().sum()}")

print("\n************************************************************************************")

# aggregating the dataset by grouping block_groups
aggregated_att = att.groupby('block_group').agg({
    'price': 'mean',
    'speed_down': 'mean',
    'speed_up': 'mean',
    'n_providers': 'sum',
    'ppl_per_sq_mile': 'mean',
    'internet_perc_broadband': 'mean',
    'lat': 'mean',
    'lon': 'mean',
    'race_perc_non_white': 'mean',
    'median_household_income': 'mean',
    'package': lambda x: x.mode()[0] # will drop this for phase 2 as we only need categorical features.
}).reset_index()

print(f"\nI am going to use this aggregated_att in phase 2 and 3:\n{aggregated_att.head()}")

print("\n************************************************************************************")

# downsampling the dataset for phase 4 for faster computation (reduce dataset size to 30% for faster computation)
att_downsampled = att.sample(frac=0.3, random_state=5805)

# creating a copy for clustering
att_clustering = att_downsampled.copy()

# dropping categorical columns for clustering
att_clustering.drop(columns=['state', 'redlining_grade', 'technology', 'package'], inplace=True)
att_clustering.drop(columns=['lat', 'lon', 'race_perc_non_white', 'internet_perc_broadband', 'block_group'], inplace=True)

print(f"\nI am using the downsampled dataset in phase 4 for faster computation.\nThe shape of the downsampled dataset is {att_c

print("\n************************************************************************************")

# encoding categrorical variables
encoded_att = pd.get_dummies(att, columns=['state', 'package', 'technology', 'redlining_grade'], drop_first=True)

# displaying the encoded dataset
print(f"\nDisplaying the encoded dataset-\n{encoded_att.head()}")

print("\n************************************************************************************")

# selecting the target variable (y)
X = encoded_att.drop(columns=['speed_up'])
y = encoded_att['speed_up']

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=5805, shuffle=True)

# num cols from original dataset
num_cols = att.select_dtypes(include=['int64', 'float64'])

# num cols from X
X_num_cols = X.select_dtypes(include=['int64', 'float64'])
```

Random Forest analysis



```python
#%% Random Forest Analysis

print("\n************************* RANDOM FOREST ANALYSIS *************************")
rf = RandomForestRegressor(random_state=5805)
rf.fit(X_train, y_train)
importances = rf.feature_importances_
feature_names = X_train.columns

sorted_indices = np.argsort(importances)[::-1]
sorted_importances = importances[sorted_indices]
sorted_features = feature_names[sorted_indices]

plt.figure(figsize=(15,15))
plt.barh(sorted_features, sorted_importances)
plt.xlabel('Feature importance')
plt.ylabel('Features')
plt.title('Random Forest Analysis')
plt.gca().invert_yaxis()
plt.grid(True)
plt.show()

rf_threshold = 0.01
selected_features = sorted_features[sorted_importances >= rf_threshold]
eliminated_features = sorted_features[sorted_importances < rf_threshold]

print(f"\nSelected Features (Random Forest): {selected_features}")
print(f"\nEliminated Features (Random Forest): {eliminated_features}")
```

PCA



```python
#%% PCA

print("\n************************* PRINCIPLE COMPONENT ANALYSIS *************************")

# standardizing for PCA
X_train_std = spl_standardized(X_train, X_train_num_cols)

pca = PCA(random_state=5805)
X_pca = pca.fit_transform(X_train_std)

cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
num_features_95 = np.argmax(cumulative_variance >= 0.95)+1
print(f"\nNumber of features needed to explain more than 95% of the variance: {num_features_95}")

plt.figure(figsize=(12,8))
plt.plot( *args: range(1, len(cumulative_variance)+1), cumulative_variance, marker='o', linestyle='--'
plt.xlabel('Number of features')
plt.ylabel('Cumulative explained variance')
plt.title('PCA - Cumulative explained variance vs Number of features')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% Variance Threshold')
plt.axvline(x=num_features_95, color='g', linestyle='--', label=f"{num_features_95} features")
plt.title('Principal Component Analysis')
plt.legend()
plt.grid()
plt.show()
```

SVD

```
print("\n************************ SINGULAR VALUE DECOMPOSITION ****************

X_std = standardized(X_num_cols)
n_components = 12
svd = TruncatedSVD(n_components=n_components, random_state=5805)
X_svd = svd.fit_transform(X_std)

feature_importance = np.abs(svd.components_)
selected_features_indices = np.argsort(-feature_importance.sum(axis=0))[:n_comp
selected_features = X.columns[selected_features_indices]
print(f"\nSelected Features:{selected_features}")
```

VIF

```
print("\n************************ VARIANCE INFLATION FACTOR ************************"

X_vif = add_constant(X_num_cols)
vif = pd.DataFrame()
vif['Variable'] = X_vif.columns
vif['VIF'] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1
print(f"\nVariance Inflation Factor Table:\n{vif}")
print("\nHere the the vif scores for fastest_speed_down and fastest_speed_price are v
```

Discretization & Binarization

```
#%% Discretization and Binarization

att['speed_category'] = pd.qcut(att['speed_down'], q=2, labels=['Slow',

plt.figure(figsize=(8, 6))
att['speed_category'].value_counts().plot(kind='bar', color=['red', 'b
plt.title('Distribution of Speed Category (Slow vs Fast)')
plt.xlabel('Speed Category')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

Anomaly

```
print("\n************************ ANOMALY/OUTLIER ANALYSIS & REMOVAL ********************

std_num_cols = standardized(num_cols)

# anomaly detection using kmeans
kmeans = KMeans(n_clusters=3, random_state=5805)
kmeans_labels = kmeans.fit_predict(std_num_cols)

# Calculate distances to cluster centers
distances_to_center = cdist(std_num_cols, kmeans.cluster_centers_, metric: 'euclidean').mi
distance_threshold = np.percentile(distances_to_center, q: 95)

# Mark anomalies
att['anomaly'] = (distances_to_center > distance_threshold).astype(int)
print(f"Number of anomalies detected: {att['anomaly'].sum()}")

# printing the shape of dataset before removal
print(f"Shape of the dataset before anomaly removal: {att.shape}")
att = att[att['anomaly'] == 0].copy()

# printing the shape of dataset after removal
print(f"Shape of the dataset after anomaly removal: {att.shape}")
```

Sample Covariance matrix

```
#%% Sample Covariance

cov_mat = num_cols.cov()

plt.figure(figsize=(18, 15))
sns.heatmap(cov_mat, annot=True, cmap='coolwarm', cbar=True)
plt.title('Sample Covariance Heatmap')
plt.xlabel('Features')
plt.ylabel('Features')
plt.show()
```

Correlation matrix

```
corr_mat = num_cols.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_mat, annot=True, cmap='coolwarm', cbar=True, vmin=-1, vmax=1, center=0
plt.title('Sample Correlation Heatmap')
plt.xlabel('Features')
plt.ylabel('Features')
plt.tight_layout()
plt.show()
```

Checking whether the target is balanced

```
#%% Checking whether the target is balanced or imbalanced

print("\n************************ CHECKING IF THE TARGET IS BALANCED ****************

target_balance = att['package'].value_counts()
print(f"\nInternet Package class value counts for comparision:\n{target_balance}")
print("\nThe target is balanced.")

print("\n************************ END OF PHASE I ************************")
```

PHASE II - Data preparation and imports

```python
#%% Importing necessary libraries
import ...
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.options.display.float_format = '{:,.3f}'.format

# loading dataset
att = pd.read_csv('speed_price_att.csv')

# dropping unnecessary columns
att.drop(columns=['collection_datetime', 'fn', 'address_full', 'incorporated_place', 'major_city', 'provider', 'speed_unit'], inp

# dropping the nan values
col_obs_todrop = ['price', 'technology', 'package']
att.dropna(subset=col_obs_todrop, how='any', inplace=True)

# dropping more unnecessary columns like categorical features as performing regression
att.drop(columns=['state', 'technology', 'package', 'redlining_grade', 'income_lmi', 'income_dollars_below_median'], inplace=True

# handling the nan values
att['n_providers'].fillna(att['n_providers'].mode(), inplace=True)
att['ppl_per_sq_mile'].fillna(att['ppl_per_sq_mile'].median(), inplace=True)
att['internet_perc_broadband'].fillna(att['internet_perc_broadband'].median(), inplace=True)

# aggregating the dataset by grouping block_group using mean for all and sum for n_providers
aggregated_att = att.groupby('block_group').agg({...}).reset_index()

# using kmeans to remove the outliers
kmeans = KMeans(n_clusters=3, random_state=5805)
kmeans.fit(aggregated_att)

# Calculate the distances of each data point from the nearest centroid
distances_to_centroid = cdist(aggregated_att, kmeans.cluster_centers_, 'euclidean').min(axis=1)

# Set a threshold for anomaly detection (e.g., 95th percentile of the distance)
threshold = np.percentile(distances_to_centroid, 95)

# Flag points as outliers (1 for outlier, 0 for non-outlier)
aggregated_att['anomaly'] = (distances_to_centroid > threshold).astype(int)

# Filter out the outliers (keep only non-outliers)
aggregated_att = aggregated_att[aggregated_att['anomaly'] == 0].copy()

# Drop the 'anomaly' column if no longer needed
aggregated_att.drop(columns=['anomaly'], inplace=True)

X = aggregated_att.drop(columns=['speed_up'])

# target as speed_up for regression
y = aggregated_att['speed_up']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5805)
```

Linear Regression

```python
#%% Linear Regression

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_train_pred = regressor.predict(X_train)
y_test_pred = regressor.predict(X_test)

plt.figure(figsize=(10, 6))
plt.scatter(y_train, y_train_pred, color='blue', label='Train Data')
plt.scatter(y_test, y_test_pred, color='red', label='Test Data')
plt.plot([min(y), max(y)], [min(y), max(y)], color='green', linestyle='--', label='Perfect Fit')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Multi-linear Regression Analysis')
plt.legend()
plt.grid(True)
plt.show()

X_train_with_const = sm.add_constant(X_train)
model = sm.OLS(y_train, X_train_with_const).fit()

r2 = r2_score(y_test, y_test_pred)
mse = mean_squared_error(y_test, y_test_pred)

# dictionary for the metrics to create a table
metrics = {
    'Metric': ['R-squared', 'Adjusted R-squared', 'AIC', 'BIC', 'MSE'],
    'Value': [
        r2,
        model.rsquared_adj,
        model.aic,
        model.bic,
        mse
    ]
}

metrics_table = pd.DataFrame(metrics)
metrics_table.set_index('Metric', inplace=True)

print("\n************************ PHASE II: REGRESSION ANALYSIS ************************")

print(f"\nMultiple Linear Regression Metrics table:\n{metrics_table}")

print("\n**************************************************************************************")
```

T-test

```python
t_test_results = model.tvalues
print(f"\nT-test results:\n{t_test_results}")

print("\n*******************************************
```

F-test

```python
f_test_result = model.fvalue
print(f"\nF-test result:\n{f_test_result}")

print("\n*******************************************
```

Confidence analysis

```python
confidence_intervals = model.conf_int(alpha=0.05)
print(f"\n95% Confidence Intervals for Coefficients:\n{confidence_int

print("\n**********************************************************
```

Stepwise regression

```python
def stepwise_regression(X_train, y_train, significance_level=0.01):
    X_train_with_const = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train_with_const).fit()
    eliminated_features = []

    while True:
        p_values = model.pvalues[1:]
        max_p_value = p_values.max()

        if max_p_value > significance_level:
            eliminated_feature = p_values.idxmax()
            print(f"Eliminating {eliminated_feature} with a p-value of {max_p_value:.4f}"
            eliminated_features.append(eliminated_feature)
            X_train_with_const = X_train_with_const.drop(columns=[eliminated_feature])
            model = sm.OLS(y_train, X_train_with_const).fit()
        else:
            break

    print(f"\nFinal Model:\n{model.summary()}")
    print()

    if eliminated_features:
        print("\nEliminated Features:")
        for feature in eliminated_features:
            print(feature)
    else:
        print("\nNo features were eliminated.")

    return model

stepwise_model = stepwise_regression(X_train, y_train)

print("\n************************ END OF PHASE II ************************")
```

PHASE III - Data Preparation

```python
#%% Importing necessary libraries and handling dataset
import ...
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.options.display.float_format = '{:,.3f}'.format

# loading the dataset
att = pd.read_csv('speed_price_att.csv')

# dropping the unnecessary columns for classification
att.drop(columns=['collection_datetime', 'fn', 'address_full', 'incorporated_place', 'major_city',
                  'provider', 'speed_unit', 'fastest_speed_down', 'fastest_speed_price',
                  'income_lmi', 'income_dollars_below_median'], inplace=True)

# dropping the na values
col_obs_todrop = ['price', 'technology', 'package']
att.dropna(subset=col_obs_todrop, how='any', inplace=True)

# handling the nan values
att['redlining_grade'].fillna(att['redlining_grade'].bfill(), inplace=True)
att['n_providers'].fillna(att['n_providers'].bfill(), inplace=True)
att['ppl_per_sq_mile'].fillna(att['ppl_per_sq_mile'].median(), inplace=True)
att['internet_perc_broadband'].fillna(att['internet_perc_broadband'].median(), inplace=True)

# merging similar packages
package_merge = {
    'AT&T FIBER INTERNET 300': 'Fiber Internet',
    'AT&T FIBER INTERNET 500': 'Fiber Internet',
    'AT&T FIBER—INTERNET 300': 'Fiber Internet',
    'AT&T FIBER — INTERNET 300': 'Fiber Internet',
    'AT&T FIBER — INTERNET 500': 'Fiber Internet',
    'AT&T FIBER—INTERNET 2000': 'Fiber Internet',
    'AT&T FIBER—INTERNET 100': 'Fiber Internet',
    'AT&T FIBER—INTERNET 500': 'Fiber Internet',
    'Internet 100': 'High-Speed Internet',
    'Internet 75': 'High-Speed Internet',
    'Internet 50': 'High-Speed Internet',
    'Internet 25': 'Basic Internet',
    'Internet 18': 'Basic Internet',
    'Internet 10': 'Basic Internet',
    'Internet Basic 5': 'Basic Internet',
    'Internet Basic 3': 'Basic Internet',
    'Internet Basic 1.5': 'Basic Internet',
    'Internet Basic 768kbps': 'Basic Internet'
}

att['package'] = att['package'].replace(package_merge)

# aggregating data by block_group
aggregated_att = att.groupby('block_group').agg({
    'price': 'mean',
    'speed_down': 'mean',
    'speed_up': 'mean',
    'n_providers': 'sum',
    'ppl_per_sq_mile': 'mean',
    'internet_perc_broadband': 'mean',
    'lat': 'mean',
    'lon': 'mean',
    'package': lambda x: x.mode()[0]
}).reset_index()

X = aggregated_att.drop(columns=['package'])

# the target is package for classification
```

```python
    prec = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')
    f1 = f1_score(y_test, y_test_pred, average='weighted')

    print(f"Confusion Matrix:\n{cm}")
    print(f"Train Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"Specificity: {weighted_specificity:.4f}")
    print(f"F1-score: {f1:.4f}")
    print(f"AUC: {auc}")

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix for {model_name}")
    plt.show()

    # making a dictionary so that I can append it in a table
    metrics = {
        'Model': model_name,
        'Train Accuracy': train_accuracy,
        'Test Accuracy': test_accuracy,
        'Precision': prec,
        'Recall': recall,
        'Specificity (Weighted)': weighted_specificity,
        'F1-score': f1,
        'AUC': auc
    }
    return metrics

metrics_list = []
```

```python
# the target is package for classification
y = aggregated_att['package']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5805, stratify=y)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

def gridsearch_and_metrics(model_name, model, param_grid, X_train, y_train, X_test, y_test, class_labels=None):

    print(f"\nStarting Grid Search for {model_name}...")
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=StratifiedKFold(n_splits=3), scoring='accura
    grid_search.fit(X_train, y_train)

    # best model using the in-built function best_estimator
    best_model = grid_search.best_estimator_

    # best parameters
    print(f"\nBest Parameters: {grid_search.best_params_}\n")

    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    # train and test accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # confusion matrix
    cm = confusion_matrix(y_test, y_test_pred)

    # calculating specificity
    specificity = {}
    weighted_specificity = 0
    total_instances = len(y_test)
    for i, class_label in enumerate(class_labels):
        tn = cm.sum() - cm[i, :].sum() - cm[:, i].sum() + cm[i, i]
        fp = cm[:, i].sum() - cm[i, i]
        specificity[class_label] = tn/(tn+fp)
        weighted_specificity += specificity[class_label]*np.sum(y_test == class_label)/total_instances

    if hasattr(best_model, "predict_proba"):
        y_test_proba = best_model.predict_proba(X_test)
        y_test_multi = label_binarize(y_test, classes=np.unique(y_test))

        plt.figure(figsize=(10, 8))

        for i, class_label in enumerate(np.unique(y_test)):
            fpr, tpr, _ = roc_curve(y_test_multi[:, i], y_test_proba[:, i])
            plt.plot(fpr, tpr, label=f"Class {class_label} (AUC = {roc_auc_score(y_test_multi[:, i], y_test_prob
        plt.plot([0, 1], [0, 1], 'k--')
        plt.title(f'Multiclass ROC Curve for {model_name}')
        plt.xlabel('False Positive Rate (FPR)')
        plt.ylabel('True Positive Rate (TPR)')
        plt.legend()
        plt.grid()
        plt.show()

        # auc score
        auc = roc_auc_score(y_test_multi, y_test_proba, multi_class='ovr')
    else:
        auc = 0

    prec = precision_score(y_test, y_test_pred, average='weighted')
```

Pre-Pruned DT

```python
print("\n************************ PRE-PRUNED DECISION TREE CLASSIFIER ************************")

pre_pruned_dt = "Pre-pruned Decision Tree"
pre_pruned_clf = DecisionTreeClassifier(random_state=5805)

param_grid_pre_pruned = {
    'max_depth': [1, 2, 3, 4, 5],
    'min_samples_split': [20,30,40],
    'min_samples_leaf': [10,20,30],
    'criterion':['gini','entropy','log_loss'],
    'splitter':['best','random'],
    'max_features':['sqrt','log2']
}

metrics = gridsearch_and_metrics(pre_pruned_dt, pre_pruned_clf, param_grid_pre_pruned, X_train, y_train, X_test, y_test, class_

metrics_list.append(metrics)
```

Post-Pruned DT

```python
print("\n************************ POST-PRUNED DECISION TREE CLASSIFIER ************************")

post_pruned_dt = "Post-pruned Decision Tree"

path = pre_pruned_clf.cost_complexity_pruning_path(X_train, y_train)
alphas = path['ccp_alphas']

accuracy_train, accuracy_test = [],[]
for a in alphas:
    post_pruned_clf = DecisionTreeClassifier(random_state=5805, ccp_alpha=a)
    post_pruned_clf.fit(X_train, y_train)
    y_train_pred_post = post_pruned_clf.predict(X_train)
    y_test_pred_post = post_pruned_clf.predict(X_test)
    accuracy_train.append(accuracy_score(y_train, y_train_pred_post))
    accuracy_test.append(accuracy_score(y_test, y_test_pred_post))

m = np.argmax(accuracy_test)
optimum_alpha = alphas[m]

best_post_pruned_clf = DecisionTreeClassifier(random_state=5805, ccp_alpha=optimum_alpha)
best_post_pruned_clf.fit(X_train, y_train)

metrics = gridsearch_and_metrics(post_pruned_dt, best_post_pruned_clf, param_grid_pre_pruned, X_train, y_train, X_test, y_test, class_l

metrics_list.append(metrics)
```

Logistic Regression

```python
print("\n************************ LOGISTIC REGRESSION CLASSIFIER ************************")

lr = "Logistic Regression"

lr_clf = LogisticRegression(random_state=5805, max_iter=1000)
param_grid_log_reg = {
    'penalty': ['l2'],
    'C': [0.1, 1, 10]
}
metrics = gridsearch_and_metrics(lr, lr_clf, param_grid_log_reg, X_train, y_train, X_test, y_test, class_lab

metrics_list.append(metrics)
```

KNN

```python
print("\n************************ K-NEAREST NEIGHBORS CLASSIFIER ************************")

knn = "K-Nearest Neighbors"

knn_clf = KNeighborsClassifier()

param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree']
}

metrics = gridsearch_and_metrics(knn, knn_clf, param_grid_knn, X_train, y_train, X_test, y_test, class_labels=y.unique())

metrics_list.append(metrics)

print("\n************************ OPTIMUM K USING ELBOW METHOD ************************")


k_values = range(2, 11)
train_accuracies = []
test_accuracies = []

for k in k_values:
    print(f"\nChecking for k = {k}")

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    train_accuracy = knn.score(X_train, y_train)
    test_accuracy = knn.score(X_test, y_test)

    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

plt.figure(figsize=(10, 6))
plt.plot(k_values, train_accuracies, label='Train Accuracy', marker='o', color='blue')
plt.plot(k_values, test_accuracies, label='Test Accuracy', marker='o', color='orange')
plt.title('Elbow Method for Optimal k (KNN) Using Accuracy')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

optimal_k = k_values[np.argmax(test_accuracies)]
print(f"Optimal k for KNN (using Elbow Method with Accuracy): {optimal_k}")
```

SVM

```python
print("\n************************ SUPPORT VECTOR MACHINE CLASSIFIER ************************")

svm = "Support Vector Machine"

svm_clf = SVC(random_state=5805, probability=True)

param_grid_svm = {
    'kernel': ['linear', 'rbf', 'poly']
}
metrics = gridsearch_and_metrics(svm, svm_clf, param_grid_svm, X_train, y_train, X_test, y_test, class_lab

metrics_list.append(metrics)
```

Naives Bayes

```python
print("\n************************ NAIVES BAYES CLASSIFIER ************************")

nb = "Naive Bayes"

nb_clf = GaussianNB()

param_grid_nb = {
    'var_smoothing': [1e-9, 1e-8, 1e-7]
}
metrics = gridsearch_and_metrics(nb, nb_clf, param_grid_nb, X_train, y_train, X_test, y_test, class_labe

metrics_list.append(metrics)
```

Neural Networks

```python
print("\n*********************** NEURAL NETWORKS CLASSIFIER ***********************")

nn = "Neural Networks"

nn_clf = MLPClassifier(random_state=5805)

param_grid_mlp = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50), (200,)],
    'activation': ['tanh', 'relu'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive']
}
metrics = gridsearch_and_metrics(nn, nn_clf, param_grid_mlp, X_train, y_train, X_test, y_test, class_labe

metrics_list.append(metrics)
```

Metrics Table

```python
#%% Display the metrics for all the classifiers

metrics_table = pd.DataFrame(metrics_list).set_index('Model')
print(metrics_table)

print("\n************************ END OF PHASE III ******************
```

PHASE IV - Data Preparation

```python
import ...
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)

# loading the dataset
att = pd.read_csv('speed_price_att.csv')

# dropping the unnecessary columns
att.drop(columns=['collection_datetime', 'fn', 'address_full', 'incorporated_place', 'major_city', 'provider', 'speed_unit'], inp

# handling missing values
att['price'].fillna(att['price'].median(), inplace=True)
att['ppl_per_sq_mile'].fillna(att['ppl_per_sq_mile'].median(), inplace=True)
att['internet_perc_broadband'].fillna(att['internet_perc_broadband'].median(), inplace=True)
att.fillna(att.mean(numeric_only=True), inplace=True)

# creating a new feature with speed_down
att['speed_category'] = pd.qcut(att['speed_down'], q=2, labels=['Slow', 'Fast'])

package_merge = {...}

att['package'] = att['package'].replace(package_merge)

# downsampling the dataset (reduce dataset size to 30% for faster computation)
att_downsampled = att.sample(frac=0.3, random_state=5805)

# creating a copy for clustering
att_clustering = att_downsampled.copy()

# dropping the categorical columns for clustering
att_clustering.drop(columns=['state', 'redlining_grade', 'technology', 'package', 'speed_category', 'income_dollars_below_median'

# standardizing the dataset for clustering
scaler = StandardScaler()
scaled_data = scaler.fit_transform(att_clustering)
```

K-Means Clustering

```python
print("\n*********************** K-MEANS CLUSTERING ***********************")

wcss = []
silhouette_scores_kmeans = []
k_values = range(2, 11)

# Calculate WCSS and silhouette scores for each k
for k in k_values:
    print(f"\nChecking for k={k}")
    kmeans = KMeans(n_clusters=k, random_state=5805, init='k-means++')
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)
    silhouette_scores_kmeans.append(silhouette_score(scaled_data, kmeans.labels_))

# Elbow Method Plot for WCSS
plt.figure(figsize=(10, 6))
plt.plot( *args: k_values, wcss, marker='o')
plt.title('Elbow Method for Optimal K (WCSS)')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()

# Silhouette Analysis Plot
plt.figure(figsize=(10, 6))
plt.plot( *args: k_values, silhouette_scores_kmeans, marker='o', color='orange')
plt.title('Silhouette Analysis for KMeans')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()

# finding optimal K using WCSS
wcss_diff = np.diff(wcss)
wcss_diff2 = np.diff(wcss_diff)

optimal_k_wcss = np.argmin(wcss_diff2) + 2
print(f"\nOptimal K (using WCSS - Elbow Method): {optimal_k_wcss}")

# finding optimal K using silhouette scores
optimal_k_silhouette = k_values[np.argmax(silhouette_scores_kmeans)]
print(f"\nOptimal K (using Silhouette Score): {optimal_k_silhouette}")
```

DBSCAN

```python
print("\n************************ DBSCAN CLUSTERING ********************

nearest_neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = nearest_neighbors.fit(scaled_data)
distances, indices = neighbors_fit.kneighbors(scaled_data)
sorted_distances = np.sort(distances[:, 4])

plt.figure(figsize=(10, 6))
plt.plot(sorted_distances)
plt.title('Nearest Neighbor Distance Analysis (for eps selection)')
plt.xlabel('Data Points')
plt.ylabel('5th Nearest Neighbor Distance')
plt.grid(True)
plt.show()

dbscan = DBSCAN(eps=1.5, min_samples=8)
dbscan_labels = dbscan.fit_predict(scaled_data)

n_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else 0)
n_noise = list(dbscan_labels).count(-1)
print(f"DBSCAN Number of clusters: {n_clusters}")
print(f"DBSCAN Number of noise points: {n_noise}")
```

Association Rule Mining

```python
#%% Apriori Algorithm for Association Rule Mining

print("\n************************ APRIORI ALGORITHM ************************")

basket = pd.get_dummies(att_downsampled[['package', 'technology', 'speed_category']])

frequent_itemsets = apriori(basket, min_support=0.3, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7, num_ite

print(f"\nFrequent Itemsets:\n{frequent_itemsets}")
print(f"\nAssociation Rules:\n{rules[['antecedents', 'consequents', 'support', 'confidence',

print("\n************************ END OF PHASE IV ************************")
```

REFERENCES

1. https://www.kaggle.com/
2. https://pandas.pydata.org/docs/index.html
3. https://numpy.org/doc/stable/index.html
4. https://scikit-learn.org/stable/index.html
5. https://docs.python.org/3/
6. https://matplotlib.org/
7. https://seaborn.pydata.org/