Checkpoint 4

December 1, 2023

1 Project title: Health Tales

1.0.1 Team members

1. Adit Doshi
2. Yash Kharade
3. Harshaditya Mallipudi
4. Prathik Makthala

The dataset we are using is collected from National Health Insurance Service in Korea. It has in total 23 columns.

1. sex - male, female
2. age - rounded to 5 years
3. height - rounded upto 5 cms
4. weight - in kilograms
5. sight_left - left eye sight
6. sight_right - right eye sight
7. hear_left - hearing left, 1(normal), 2(abnormal)
8. hear_right - hearing right, 1(normal), 2(abnormal)
9. SBP - Systolic blood pressure
10. DBP - Diastolic blood pressure
11. BLDS - BLDS or FSG(fasting blood glucose)
12. tot_chole - total cholesterol
13. HDL_chole - HDL cholesterol
14. LDL_chole - LDL cholesterol
15. triglyceride
16. hemoglobin
17. urine_protein - protein in urine, 1(-), 2(+/-), 3(+1), 4(+2), 5(+3), 6(+4)
18. serum_creatinine - serum(blood) creatinine
19. SGOT_AST - SGOT(Glutamate-oxaloacetate transaminase) AST(Aspartate transaminase)
20. SGOT_ALT - ALT(Alanine transaminase)
21. gamma_GTP - y-glutamyl transpeptidase
22. SMK_stat_type_cd - Smoking state, 1(never), 2(used to smoke but quit), 3(still smoke)
23. DRK_YN - Drinker or Not

1.1 Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪roc_auc_score, roc_curve
```

1.2 Loading data from CSV

[2]:
```
df = pd.read_csv("smoking_driking_dataset_Ver01.csv")
```

1.3 Data Cleaning

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 991346 entries, 0 to 991345
Data columns (total 24 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   sex               991346 non-null  object
 1   age               991346 non-null  int64
 2   height            991346 non-null  int64
 3   weight            991346 non-null  int64
 4   waistline         991346 non-null  float64
 5   sight_left        991346 non-null  float64
 6   sight_right       991346 non-null  float64
 7   hear_left         991346 non-null  float64
 8   hear_right        991346 non-null  float64
 9   SBP               991346 non-null  float64
 10  DBP               991346 non-null  float64
 11  BLDS              991346 non-null  float64
 12  tot_chole         991346 non-null  float64
 13  HDL_chole         991346 non-null  float64
 14  LDL_chole         991346 non-null  float64
 15  triglyceride      991346 non-null  float64
 16  hemoglobin        991346 non-null  float64
 17  urine_protein     991346 non-null  float64
 18  serum_creatinine  991346 non-null  float64
 19  SGOT_AST          991346 non-null  float64
 20  SGOT_ALT          991346 non-null  float64
 21  gamma_GTP         991346 non-null  float64
 22  SMK_stat_type_cd  991346 non-null  float64
 23  DRK_YN            991346 non-null  object
dtypes: float64(19), int64(3), object(2)
memory usage: 181.5+ MB
```

Apart from sex and the DRN_YK(target variable) feature all of the features are quantative.

There is no null value in any of the columns.

[4]:
```python
df.nunique()
```

[4]:
```
sex                    2
age                   14
height                13
weight                24
waistline            737
sight_left            24
sight_right           24
hear_left              2
hear_right             2
SBP                  171
DBP                  127
BLDS                 498
tot_chole            474
HDL_chole            223
LDL_chole            432
triglyceride        1657
hemoglobin           190
urine_protein          6
serum_creatinine     183
SGOT_AST             568
SGOT_ALT             594
gamma_GTP            940
SMK_stat_type_cd       3
DRK_YN                 2
dtype: int64
```

[5]:
```python
df.drop_duplicates(inplace=True)
```

[6]:
```python
df.dropna(inplace=True)
```

[7]:
```python
df_for_eda = pd.get_dummies(df, drop_first=True)
```

1.4 Exploratory Data Analysis

Below are the min, max, mean and other values for all the features:

[8]:
```
df_for_eda.describe().transpose()
```

[8]:
```
                     count        mean         std    min    25%    50%  \
age               991320.0   47.614529   14.181346   20.0   35.0   45.0
height            991320.0  162.240563    9.282922  130.0  155.0  160.0
weight            991320.0   63.283884   12.514101   25.0   55.0   60.0
waistline         991320.0   81.233255   11.850296    8.0   74.1   81.0
sight_left        991320.0    0.980833    0.605954    0.1    0.7    1.0
sight_right       991320.0    0.978428    0.604779    0.1    0.7    1.0
hear_left         991320.0    1.031495    0.174652    1.0    1.0    1.0
hear_right        991320.0    1.030476    0.171892    1.0    1.0    1.0
SBP               991320.0  122.432360   14.543083   67.0  112.0  120.0
DBP               991320.0   76.052549    9.889334   32.0   70.0   76.0
BLDS              991320.0  100.424305   24.179852   25.0   88.0   96.0
tot_chole         991320.0  195.556769   38.660092   30.0  169.0  193.0
HDL_chole         991320.0   56.936984   17.238578    1.0   46.0   55.0
LDL_chole         991320.0  113.037429   35.842938    1.0   89.0  111.0
triglyceride      991320.0  132.140030  102.194762    1.0   73.0  106.0
hemoglobin        991320.0   14.229810    1.584924    1.0   13.2   14.3
urine_protein     991320.0    1.094221    0.437719    1.0    1.0    1.0
serum_creatinine  991320.0    0.860467    0.480536    0.1    0.7    0.8
SGOT_AST          991320.0   25.989424   23.493668    1.0   19.0   23.0
SGOT_ALT          991320.0   25.755148   26.308910    1.0   15.0   20.0
gamma_GTP         991320.0   37.136152   50.423811    1.0   16.0   23.0
SMK_stat_type_cd  991320.0    1.608112    0.818504    1.0    1.0    1.0

                    75%     max
age                60.0    85.0
height            170.0   190.0
weight             70.0   140.0
waistline          87.8   999.0
sight_left          1.2     9.9
sight_right         1.2     9.9
hear_left           1.0     2.0
hear_right          1.0     2.0
SBP               131.0   273.0
DBP                82.0   185.0
BLDS              105.0   852.0
tot_chole         219.0  2344.0
HDL_chole          66.0  8110.0
LDL_chole         135.0  5119.0
triglyceride      159.0  9490.0
hemoglobin         15.4    25.0
urine_protein       1.0     6.0
serum_creatinine    1.0    98.0
SGOT_AST           28.0  9999.0
SGOT_ALT           29.0  7210.0
```

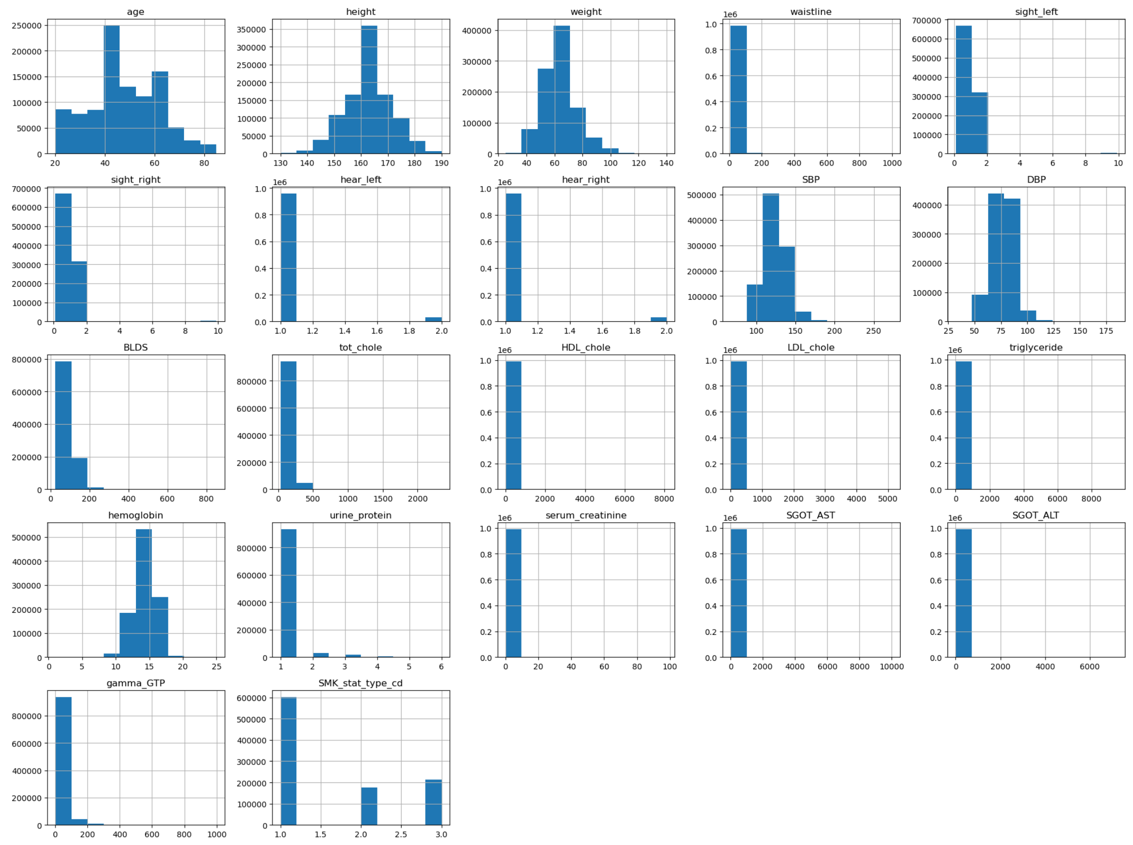
```
gamma_GTP          39.0   999.0
SMK_stat_type_cd    2.0     3.0
```

Observations:

1. waistline: The maximum value of 999 cm in the waistline column appears to be an outlier or potentially an error, as it is significantly higher than the other values.
2. HDL_chole: The extremely high maximum value of 8110 mg/dL for HDL cholesterol seems unusual and could be an outlier.
3. serum_creatinine: The maximum value of 98 mg/dL is considerably higher than the other values in the serum creatinine column.
4. SGOT_AST and SGOT_ALT: Both SGOT and SGOT levels have very high maximum values (9999 and 7210, respectively), which could be outliers or data entry errors.
5. gamma_GTP: The maximum value of 999 is significantly higher than the other values, indicating a potential outlier.
6. BLDS: The maximum value of 852 for blood sugar appears to be significantly higher than the typical range.

Histogram showing frequency of all the features:

[9]:
```
df_for_eda.hist(figsize=(20,15))
plt.tight_layout()
plt.show()
```



[10]:
```
df_for_eda.skew()
```

[10]:
```
age                  0.153652
height              -0.022721
weight               0.576537
waistline           26.789317
sight_left           9.994631
sight_right         10.033672
hear_left            5.365007
hear_right           5.463034
SBP                  0.482035
DBP                  0.400013
BLDS                 4.617495
tot_chole            1.556927
HDL_chole          104.578552
LDL_chole            5.251830
triglyceride         6.529776
hemoglobin          -0.383984
urine_protein        5.672664
serum_creatinine   111.021200
```

```
SGOT_AST            150.490203
SGOT_ALT             50.038404
gamma_GTP             7.718657
SMK_stat_type_cd      0.831452
sex_Male             -0.124272
DRK_YN_Y              0.000742
dtype: float64
```

Symmetric Distributions (Skewness 0): age: The distribution of age appears to be approximately symmetric with a skewness close to 0, indicating that it's balanced around the mean.

height: Similarly, the height distribution is very close to symmetric.

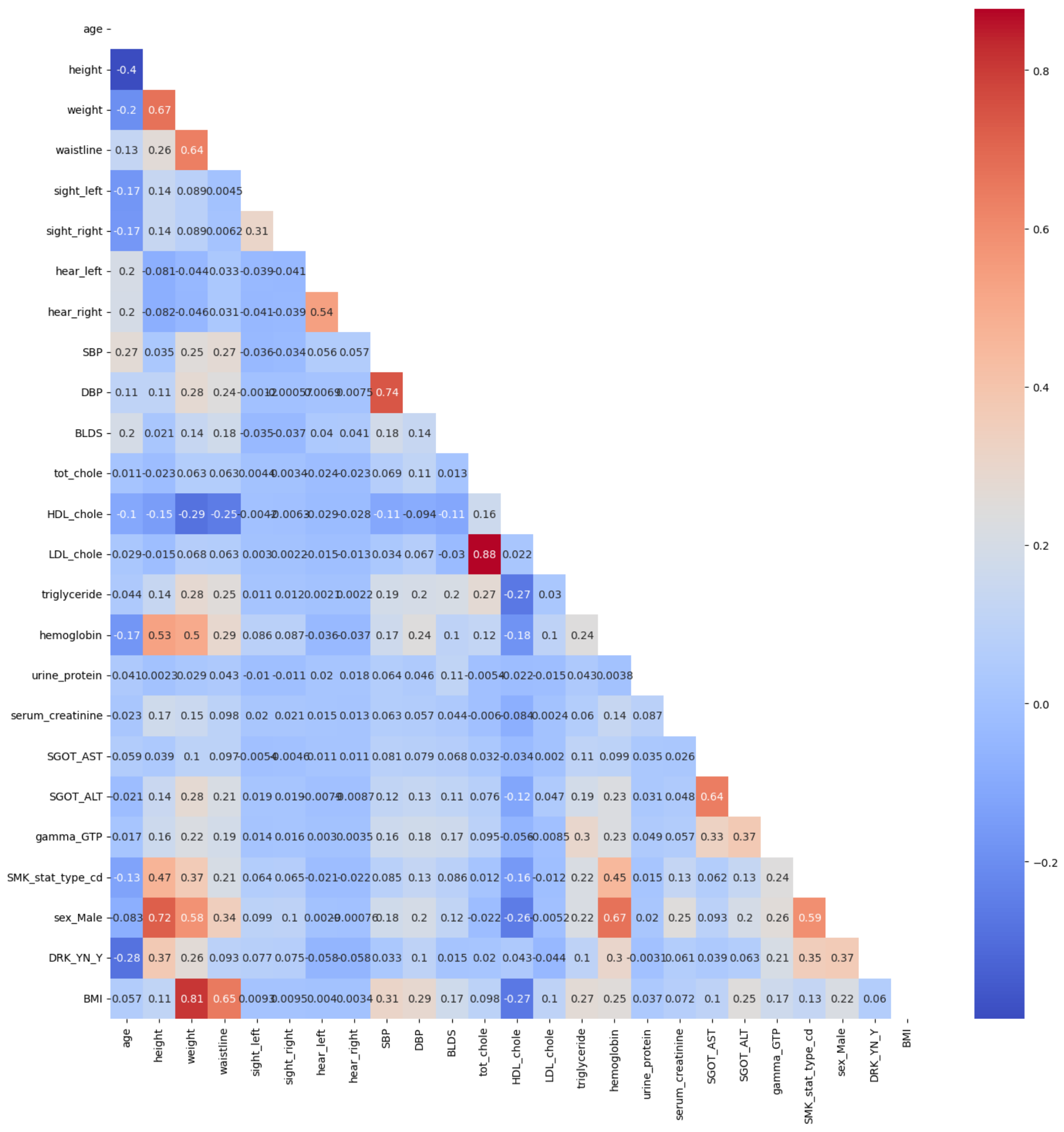
Right-Skewed Distributions (Positive Skewness): weight, SBP (Systolic Blood Pressure), DBP (Diastolic Blood Pressure), tot_chole (Total Cholesterol), LDL_chole (LDL Cholesterol), triglyceride, hemoglobin: These columns exhibit right-skewed distributions, as indicated by positive skewness values. This means that the majority of data points are concentrated on the left side of the distribution, with a long tail on the right.

Strongly Right-Skewed Distributions: waistline, sight_left, sight_right, hear_left, hear_right, BLDS (Blood Sugar), HDL_chole (HDL Cholesterol), serum_creatinine, SGOT_AST (Aspartate Aminotransferase), gamma_GTP: These columns have very high positive skewness values, indicating that they are strongly right-skewed. These variables have long right tails with most values concentrated on the lower end.

Left-Skewed Distribution (Negative Skewness): sex_Male: This column has a slightly negative skewness, suggesting that there may be more females than males in the dataset. A negative skew means that the data is concentrated on the right side with a long tail on the left.

[11]:
```
df_for_eda['BMI'] = df_for_eda['weight'] / (df_for_eda['height']/100)**2
```

[12]:
```
# Visualize correlation matrix
corr_matrix = df_for_eda.corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

plt.figure(figsize=(17, 17))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm",mask = mask)
plt.show()
```



None of the features have a strong coorelation with the target variable

Splitting the data:

```
[13]: X = df.drop("DRK_YN", axis=1)

y = df["DRK_YN"].map({'Y': 1, 'N': 0})

X_encoded = pd.get_dummies(X, drop_first=True)

df_encoded = pd.concat([X_encoded, y], axis=1)
```

```
# Computing the correlation matrix
corr_matrix = df_encoded.corr()

correlations_with_target = corr_matrix["DRK_YN"].drop("DRK_YN")

threshold = 0.2
selected_features = correlations_with_target[correlations_with_target.abs() >␣
↪threshold].index.tolist()

X_selected = X_encoded[selected_features]

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
↪3, random_state=42)

# Normalize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

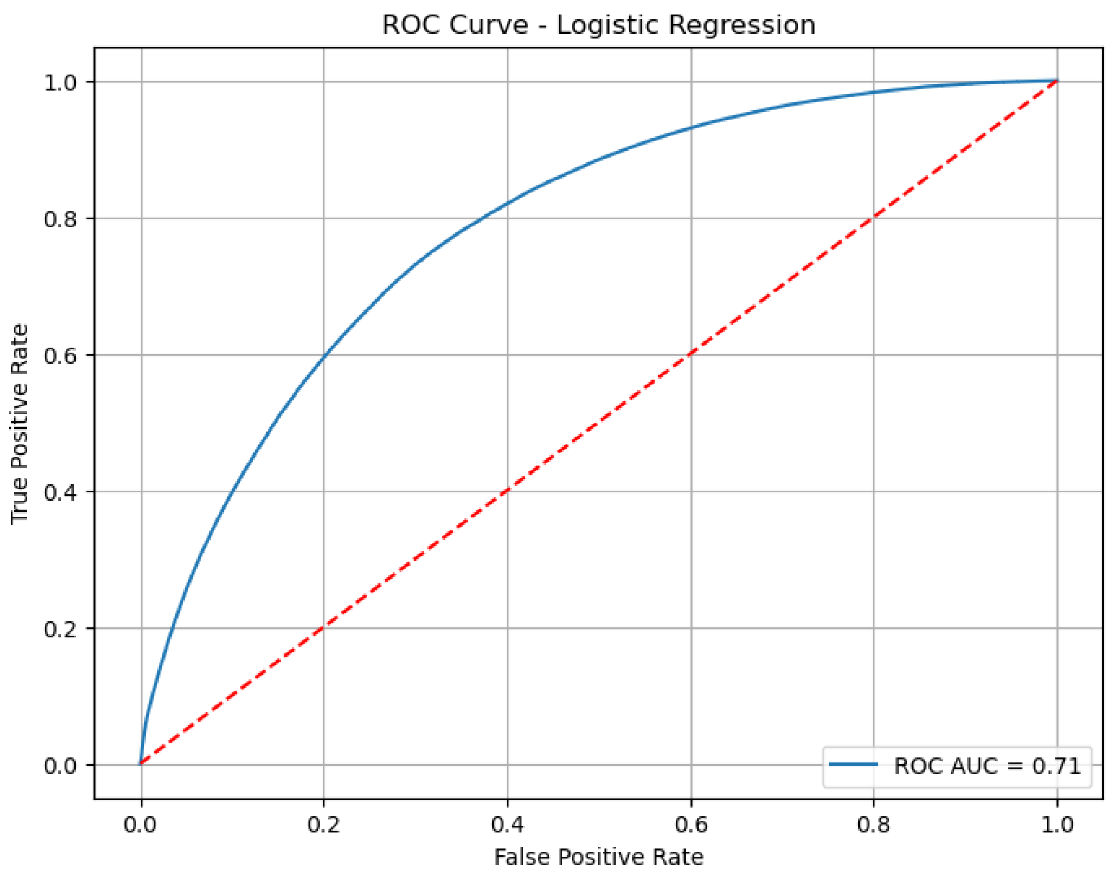
1.4.1 Model Training

For priliminary analysis, we have selected Logistic Regression, Random forest and K-nearest neighbours models for our data.

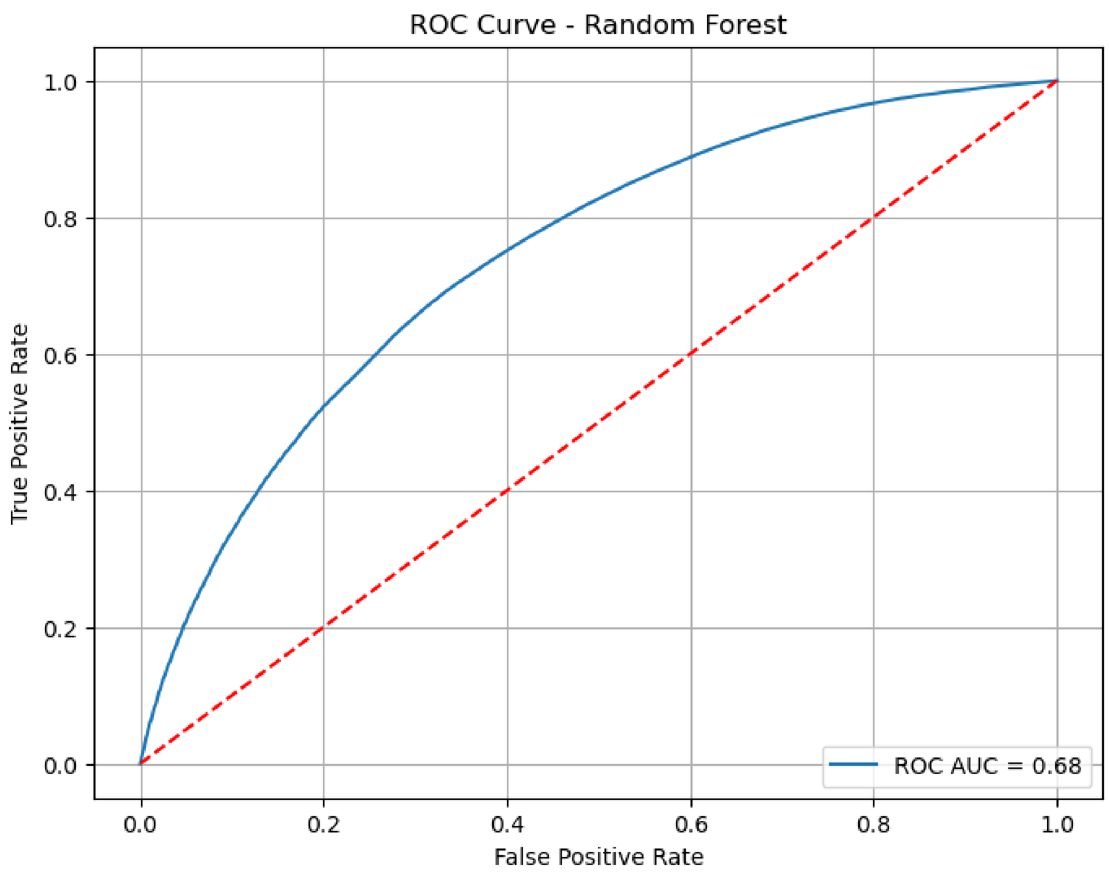
Below are some results and plotted ROC curves for all three algorithms

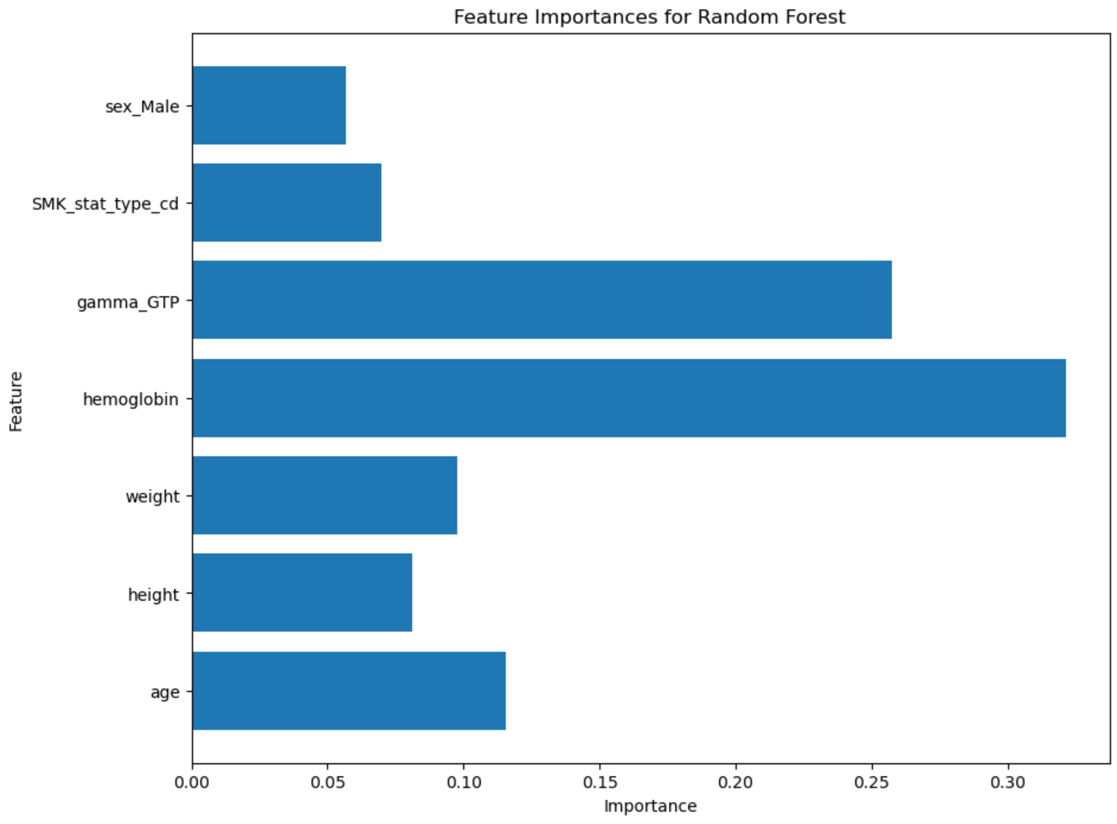
[31]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve,␣
↪classification_report
import matplotlib.pyplot as plt

models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
    "K-Nearest Neighbors": KNeighborsClassifier()
}

original_model_accuracies = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred_test = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred_test)
    original_model_accuracies[name] = accuracy  # Store each model's accuracy␣
↪in the dictionary

    print(f"Model: {name}")
    print(classification_report(y_test, y_pred_test))
    print("The TEST accuracy is", accuracy)

    # Compute and display ROC AUC score
    roc_score = roc_auc_score(y_test, y_pred_test)
    print("The ROC score for TEST data is", roc_score)

    # Plot ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'ROC AUC = {roc_score:.2f}')
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {name}')
    plt.legend(loc='lower right')
    plt.grid()
    plt.show()
    if name == "Random Forest":
        # Using feature_importances_ for RandomForestClassifier
        feature_names = X_selected.columns
        plt.figure(figsize=(10, 8))
        plt.barh(feature_names, model.feature_importances_)
        plt.xlabel('Importance')
        plt.ylabel('Feature')
        plt.title(f'Feature Importances for {name}')
        plt.show()
```

```
Model: Logistic Regression
              precision    recall  f1-score   support

           0       0.71      0.72      0.72    148796
           1       0.72      0.70      0.71    148600

    accuracy                           0.71    297396
   macro avg       0.71      0.71      0.71    297396
weighted avg       0.71      0.71      0.71    297396

The TEST accuracy is 0.7129416670029187
The ROC score for TEST data is 0.7129337993246248
```

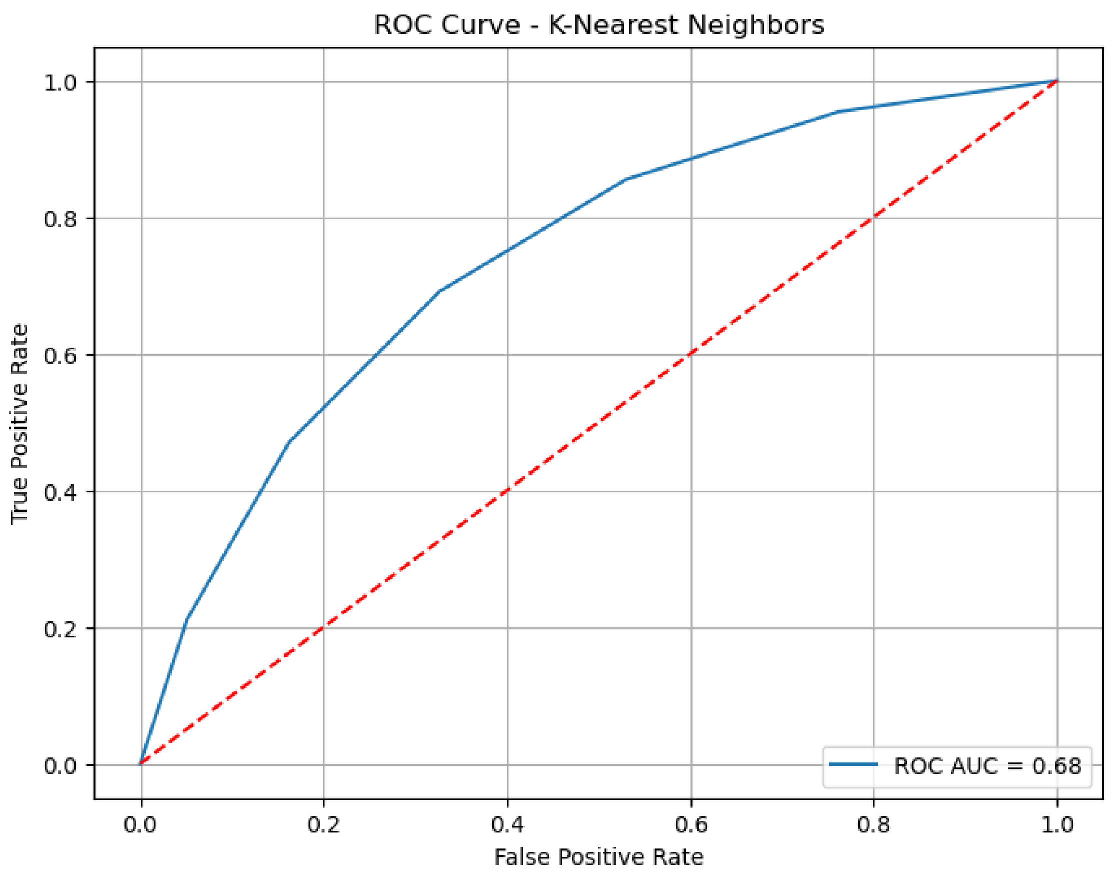


```
Model: Random Forest
              precision    recall  f1-score   support

           0       0.68      0.67      0.68    148796
           1       0.67      0.69      0.68    148600

    accuracy                           0.68    297396
   macro avg       0.68      0.68      0.68    297396
weighted avg       0.68      0.68      0.68    297396

The TEST accuracy is 0.6786607755316144
The ROC score for TEST data is 0.6786680265938638
```


ROC Curve - Random Forest
True Positive Rate
False Positive Rate
ROC AUC = 0.68
0.0
0.2
0.4
0.6
0.8
1.0

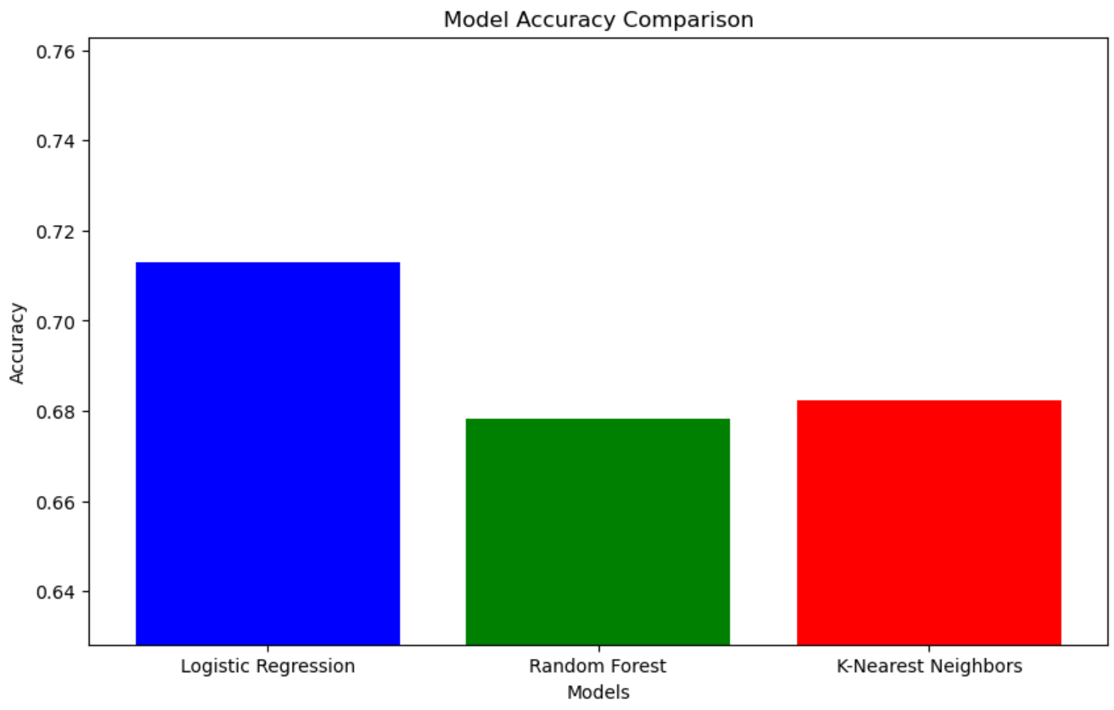


```
Model: K-Nearest Neighbors
              precision    recall  f1-score   support

           0       0.69      0.67      0.68    148796
           1       0.68      0.69      0.68    148600

    accuracy                           0.68    297396
   macro avg       0.68      0.68      0.68    297396
weighted avg       0.68      0.68      0.68    297396

The TEST accuracy is 0.682315834779217
The ROC score for TEST data is 0.682321551683559
```



Accuracy comparison for all the three models. Without any fine tuning, Logistic regression gave us the best accuracy followed by KNN and random forest.

[15]:
```python
plt.figure(figsize=(10, 6))
plt.bar(models.keys(), model_accuracies, color=['blue', 'green', 'red'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim([min(model_accuracies) - 0.05, max(model_accuracies) + 0.05])
plt.show()
```



General Observations:

1. The Logistic Regression model has the highest accuracy, followed by K-Nearest Neighbors and Random Forest.
2. Precision, recall, and F1-scores are similar for all three models, suggesting that they perform similarly in terms of correctly classifying instances from both classes.
3. The ROC scores for all models are also similar, indicating comparable discrimination abilities.

1.5 Set Up Logistic Regression and GridSearchCV

[22]:
```python
import warnings
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings('ignore')

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

C_values = [0.001, 0.01, 0.1, 1, 10, 100]
solvers = ['lbfgs', 'newton-cg', 'sag', 'saga']
max_iters = [100, 200, 300, 500]  # Increased max_iter
```

```
results = []

for C in C_values:
    for solver in solvers:
        for max_iter in max_iters:
            try:
                logreg = LogisticRegression(C=C, solver=solver,␣
↪max_iter=max_iter)
                logreg.fit(X_train_scaled, y_train)
                y_pred = logreg.predict(X_test_scaled)
                accuracy = accuracy_score(y_test, y_pred)

                results.append({'C': C, 'solver': solver, 'max_iter': max_iter,␣
↪'accuracy': accuracy})
            except ValueError:
                continue

sorted_results = sorted(results, key=lambda x: x['accuracy'], reverse=True)

for result in sorted_results[:5]:
    print(result)
```

```
{'C': 1, 'solver': 'sag', 'max_iter': 200, 'accuracy': 0.7129618421229607}
{'C': 10, 'solver': 'sag', 'max_iter': 200, 'accuracy': 0.7129584796029537}
{'C': 100, 'solver': 'sag', 'max_iter': 200, 'accuracy': 0.7129584796029537}
{'C': 0.1, 'solver': 'sag', 'max_iter': 100, 'accuracy': 0.7129551170829467}
{'C': 1, 'solver': 'saga', 'max_iter': 200, 'accuracy': 0.7129551170829467}
```

[21]:

```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

warnings.filterwarnings('ignore')

param_grid = {
    'max_depth': [None, 10, 20],
    'max_features': ['auto'],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [2, 5],
    'n_estimators': [10, 50, 100]
}

rf = RandomForestClassifier()
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,␣
↪n_jobs=-1, verbose=2)
```

```
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)

best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)

best_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the best model:", best_accuracy)

print("\nAccuracies for all parameter combinations:")
for i, params in enumerate(grid_search.cv_results_['params']):
    print(f"Parameters: {params} - Accuracy: {grid_search.
 ↪cv_results_['mean_test_score'][i]}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf':
1, 'min_samples_split': 5, 'n_estimators': 100}
Accuracy of the best model: 0.721761556981264

Accuracies for all parameter combinations:
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.667940582308622
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.6770727067508013
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.6784402921222288
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.6843804218137597
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.6918653918394967
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.6929130577866849
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.691052624823594
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.6976859139425777
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.6987753701424188
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.6933021489202353
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.6995391427516857
Parameters: {'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.7001905110023451
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.7203872473795088
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.7211827236200854
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.7214867915194109
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.720529914395336
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.7211711956716093
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.7216107242418389
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.7206394371588057
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.7215487583375035
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.7214363535324007
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.7207907491573353
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.7214291481635365
Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.7214939973036193
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.7039776693063252
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.7104740574863995
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.711479930720021
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.7074463484748386
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.7128936314261616
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.7137049579560032
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 10} - Accuracy: 0.7087188237311077
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50} - Accuracy: 0.7137539555504049
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100} - Accuracy: 0.7141661037438711
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 10} - Accuracy: 0.7089955087719887
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 50} - Accuracy: 0.7140464950859101
Parameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 100} - Accuracy: 0.7147526248792502
```

```
[23]: from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Initial Decision Tree Accuracy:", accuracy_dt)
```

```
Initial Decision Tree Accuracy: 0.637678381686371
```

```
[24]: from sklearn.model_selection import GridSearchCV

param_grid_dt = {
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'auto', 'sqrt', 'log2']
}

grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt, cv=5,␣
  ↪n_jobs=-1, verbose=2)
grid_search_dt.fit(X_train_scaled, y_train)

print("Best Parameters for Decision Tree:", grid_search_dt.best_params_)

best_dt = grid_search_dt.best_estimator_
y_pred_best_dt = best_dt.predict(X_test_scaled)

best_accuracy_dt = accuracy_score(y_test, y_pred_best_dt)
print("Best Accuracy for Decision Tree:", best_accuracy_dt)
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
Best Parameters for Decision Tree: {'max_depth': 10, 'max_features': None,
'min_samples_leaf': 2, 'min_samples_split': 2}
Best Accuracy for Decision Tree: 0.7201643599779418
```

```
[28]: print("Best Accuracy for Logistic Regression:", sorted_results[:1])
print("Best Accuracy for Random Forest:", best_accuracy)
print("Best Accuracy for Decision Tree:", best_accuracy_dt)

model_accuracies = {
    "Logistic Regression": sorted_results[:1],
    "Random Forest": best_accuracy,
    "Decision Tree": best_accuracy_dt
}
```

```
sorted_accuracies = sorted(model_accuracies.items(), key=lambda x: x[1],␣
↪reverse=True)
for model, accuracy in sorted_accuracies:
    print(f"Model: {model} - Accuracy: {accuracy}")
```

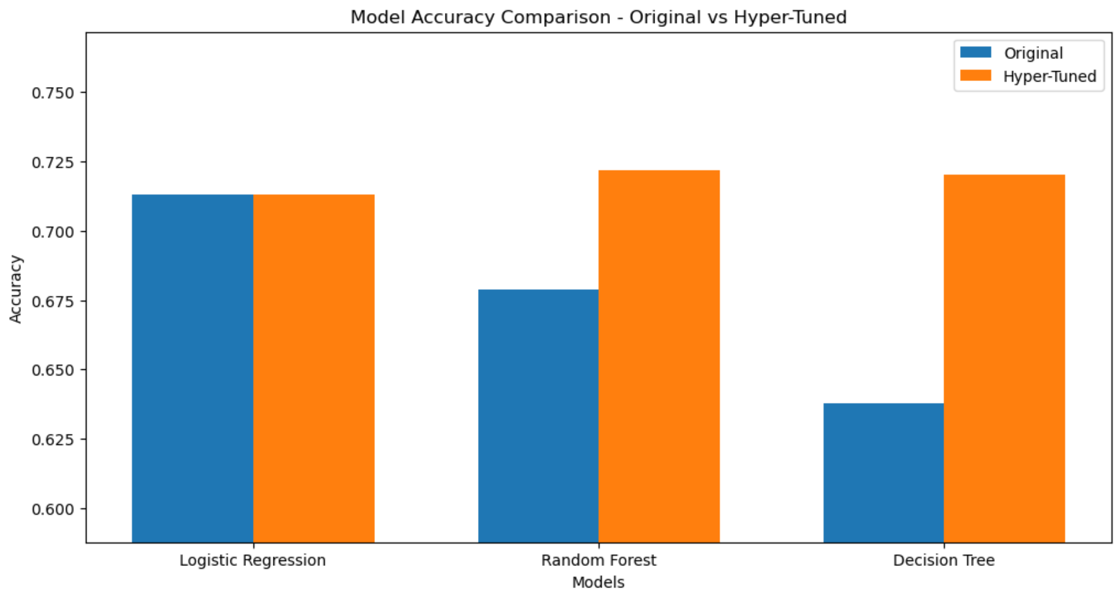
```
Best Accuracy for Logistic Regression: [{'C': 1, 'solver': 'sag', 'max_iter':
200, 'accuracy': 0.7129618421229607}]
Best Accuracy for Random Forest: 0.721761556981264
Best Accuracy for Decision Tree: 0.7201643599779418
```

[37]:
```
print(original_model_accuracies)
accuracy_lr = original_model_accuracies['Logistic Regression']
accuracy_rf = original_model_accuracies['Random Forest']
accuracy_knn = original_model_accuracies['K-Nearest Neighbors']

print("Accuracy for Logistic Regression:", accuracy_lr)
print("Accuracy for Random Forest:", accuracy_rf)
print("Accuracy for K-Nearest Neighbors:", accuracy_knn)
```

```
{'Logistic Regression': 0.7129416670029187, 'Random Forest': 0.6786607755316144,
'K-Nearest Neighbors': 0.682315834779217}
Accuracy for Logistic Regression: 0.7129416670029187
Accuracy for Random Forest: 0.6786607755316144
Accuracy for K-Nearest Neighbors: 0.682315834779217
```

[39]:
```
import matplotlib.pyplot as plt
import numpy as np


original_accuracy_lr = accuracy_lr
original_accuracy_rf = accuracy_rf
original_accuracy_knn = accuracy_knn


best_accuracy_lr = sorted_results[0]['accuracy']
best_accuracy_rf = best_accuracy

original_model_accuracies = {
    "Logistic Regression": original_accuracy_lr,
    "Random Forest": original_accuracy_rf,
    "Decision Tree": accuracy_dt
}

hyper_tuned_accuracies = {
    "Logistic Regression": best_accuracy_lr,
    "Random Forest": best_accuracy_rf,
    "Decision Tree": best_accuracy_dt
}
```

```
labels = original_model_accuracies.keys()

original_scores = original_model_accuracies.values()
tuned_scores = hyper_tuned_accuracies.values()

x = np.arange(len(labels))
width = 0.35

plt.figure(figsize=(12, 6))
plt.bar(x - width/2, original_scores, width, label='Original')
plt.bar(x + width/2, tuned_scores, width, label='Hyper-Tuned')

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison - Original vs Hyper-Tuned')
plt.xticks(x, labels)
plt.ylim([min(min(original_scores), min(tuned_scores)) - 0.05,␣
 ↪max(max(original_scores), max(tuned_scores)) + 0.05])
plt.legend()

plt.show()
```



```
[ ]:
```