Musikpreferensanalys med Maskininlärning

Kurs: Pythonprogrammering för AI-utveckling

Författare: Fredrik Barré
Datum: 2025-03-11

Innehåll

1. Introduktion

1.1 Bakgrund

Jag ville utgå från exemplet från Youtube, baserat på kopplingen mellan demografi och kulturella preferenser, när det gäller musik genrer. Detta bara för att under rådande förutsättningar lösa problemet från ett Python, OOP samt maskininlärningssynvinkel och visa på min förmåga att ta till mig en problemställning och skapa en lösning från dessa förutsättningar. Formen och omfattningen samt att kunna implementera dessa i ett projekt var det som sporrade mig att komma fram med denna lösning.

Jag insåg snart att det även behövs mer data för att kunna, syntetiskt, få ett lite bättre underlag för att träna modellen med, varför jag även skapade ett program för att skapa just syntetisk data, åtminstone 2500 rader indata. Men självkart går det att via tex kaggle och andra källor hämta underlag med betydligt mer data.

1.2 Problemformulering

I mitt projekt har jag siktat på att utveckla och implementera en maskininlärningsmodell som kan förutsäga musikgenrepreferenser baserat på demografiska faktorer, primärt ålder och kön. Målet var att skapa ett verktyg som kan hjälpa till att:

1. **Analysera sambandet mellan demografiska faktorer och musikpreferenser**
2. **Förutsäga vilken musikgenre en person med specifika demografiska egenskaper sannolikt föredrar**
3. **Visualisera trender och mönster i hur musiksmak varierar över olika åldersgrupper och mellan könen**

Jag ville demonstrera hela maskininlärningsprocessen från datainsamling och analys till modellträning, utvärdering och implementation i en användarvänlig programapplikation.

1.3 Projektets Omfattning och Avgränsningar

Mitt fokus har varit på att utveckla en klassificeringsmodell för att förutsäga musikgenrepreferenser hos individer baserat på deras demografiska egenskaper. Jag har utvecklat en fullständig applikation som tillåter användare att utforska data, träna olika maskininlärningsmodeller, och göra prediktioner baserat på den tränade modellen.

Projektet omfattar följande huvudområden:

- **Datautforskning och visualisering**
- **Databehandling och förbehandling**
- **Modellval, träning och optimering**
- **Utvärdering av modellprestanda**
- **Utveckling av ett användargränssnitt för interaktion med modellen**

För att projektet skulle vara genomförbart inom tidsramen gjorde jag följande avgränsningar: Data-avgränsningar:

- **Endast grundläggande demografiska faktorer (ålder och kön) används som ingångsvariabler**
- **Ett begränsat antal musikgenrer analyseras**
- **Kulturella och regionala skillnader beaktas inte i denna version Tekniska avgränsningar:**
- **Fokus på klassificeringsmodeller (inte regressionsmodeller eller rekommendationssystem)**
- **Huvudsakliga utvärderingsmetriker begränsas till precision, recall och F1-score**
- **Ingen integration med externa musikplattformar eller API:er Funktionella avgränsningar:**
- **Applikationen fungerar offline med lokalt lagrad data**
- **Ingen kontinuerlig inlärning eller modelluppdatering i realtid**
- **Inga personliga användardata samlas in**

Dataförståelse

- **Datakomplett: Grund datat, mitt dataset, i mitt grundresultat är inte på långa vägar komplett. Standarddatasetet för musikpreferenser innehåller endast 18 kompletta poster. Vid testning med externa dataset utför applikationen automatiska kontroller av fullständighet.**
- **Null-värden: Jag har även implementerat en metod för att upptäcka och hantera null-värden. Applikationen kontrollerar systematiskt olika former av null-värden inklusive tomma celler, 'N/A'-strängar och NaN-värden, och erbjuder flera imputeringsstrategier för att bevara dataintegritet.**
- **Extrema värden: Programmet letar även efter så kallade ”outliers” i datasetet. Åldersoutliers identifieras med IQR-metoden (värden bortom Q1-1,5×IQR eller Q3+1,5×IQR). I standarddatasetet varierar åldersvärden från 20-37 år, utan upptäckta outliers.**
- **Datatyper: Programmet identifierar olika datatyper i dataset och öppnar för att hantera eventuella överväganden kring konvertering. Datasetet innehåller två primära feature-datatyper: ålder (kontinuerlig numerisk) och kön (binär kategorisk). Målvariabeln 'genre' är kategorisk med 5 distinkta klasser.**
- **Val av fält: “Efter analys valdes ålder och kön som primära features på grund av deras starka korrelation med musikpreferenser, vilket visualiseras i ålder-pergenre-boxplot och kön-genre-frekvensfördelningar.**
- **Numerisk konvertering: Kön kodades som en binär feature (0=kvinna, 1=man). Målvariabeln 'genre' behålls som kategoriska etiketter för klassificering. Åldersvärden standardiserades med StandardScaler för att säkerställa lika viktning i avståndsbaserade algoritmer som KNN.**

2. Metodologi

2.1 Val av Dataset

För mitt projekt valde jag att använda ett anpassat dataset som innehåller information om individers ålder, kön och favoritmusikmusiksgenre. Datasetet representerar ett tvärsnitt av befolkningen med varierande åldrar och en någorlunda jämn könsfördelning. Jag integrerade ett defaultdataset med 18 datapunkter i applikationen, men gav även användaren möjlighet att ladda in egna dataset i CSV-format.

Defaultdatasetets struktur:

- **18 rader representerar olika individer**
- **Åldrar varierar från 20-37 år**
- **Könsfördelning: 9 män (kodade som 1) och 9 kvinnor (kodade som 0)**
- **Genrer inkluderar: HipHop, Jazz, Classical, Dance, och Acoustic Jag valde detta dataset eftersom det:**
- **Innehåller de demografiska faktorer jag ville analysera (ålder och kön)**
- **Har en tydlig målvariabel (musikgenre) för klassificering**
- **Är tillräckligt litet för snabb bearbetning men tillräckligt stort för att identifiera mönster**

2.2 Dataanalys

2.2.1 Datakvalitet

För att säkerställa en robust modell började jag med en grundlig analys av datakvaliteten. Denna process byggde jag in i applikationen och omfattar flera viktiga steg:

Komplett data vs saknad data: Jag implementerade en systematisk genomgång av alla datapunkter för att identifiera eventuella luckor. Om saknade värden identifieras, erbjuder min applikation användaren flera alternativ:

- **Imputering med medelvärde**
- **Imputering med median**
- **Imputering med typvärde**
- **Borttagning av rader med saknade värden**

Null-värden och deras hantering: Min applikation har stöd för att identifiera och hantera olika typer av null-värden som kan förekomma i dataset:

- **Tomma celler i CSV-filer**
- **“Unknown”, “N/A” eller andra textrepresentationer av saknade värden**
- **Numeriska specialvärden (t.ex. NaN, Inf)**

För varje typ av null-värde implementerade jag lämpliga hanteringsstrategier baserat på användarens val.

Extremvärden och deras påverkan: En viktig del av min dataanalys är identifiering och hantering av extremvärden:

- **Jag använder boxplot-analys för att identifiera outliers i ålder**
- **Kvartilmetoden (1.5 * IQR) tillämpas för matematisk detektion av avvikande värden**
- **Användaren kan välja mellan olika strategier för att hantera outliers:**
    - **Begränsning ("capping") till gränsvärden**
    - **Ersättning med medelvärde/median**
    - **Borttagning av extrema datapunkter**

2.2.2 Datatyper och Struktur

I mitt projekt arbetar jag med följande datatyper:

Numeriska variabler:

- **Ålder (kontinuerlig variabel)**
- **Kön (binär kategorisk variabel kodad som 0/1) Kategoriska variabler:**
- **Musikgenrer (målvariabel med olika klasser) Dessa datatyper hanterar jag på följande sätt:**
- **Ålder standardiseras för att fungera bättre med maskininlärningsmodeller**
- **Kön används som binär feature direkt**
- **Musikgenrer (målvariabel) behålls som kategoriska etiketter**

För databearbetning och analys i applikationen använder jag primärt pandas-biblioteket för dataramar och numpy för numeriska operationer. För visualisering valde jag matplotlib och seaborn för att skapa informativa diagram över datafördelningar och relationer.

2.2.3 Dataförbehandling

Dataförbehandlingen är en kritisk del av projektet och jag har implementerat den i applikationskoden genom följande steg:

Normalisering: För att modeller ska prestera optimalt normaliserar jag numeriska variabler:

- **Ålder standardiseras med StandardScaler (meancentrering och skalning till enhetsvarians)**
- **Denna standardisering säkerställer att modeller som är känsliga för skalning (som KNN) fungerar korrekt**

Databalansering: Jag insåg att obalanserade klasser i träningsdata kan leda till partiska modeller. Min applikation analyserar:

- **Fördelningen av målvariabeln (musikgenrer)**
- **Visar fördelningen genom visualiseringar**
- **Användaren får insikt i eventuell obalans före träning**

Feature engineering: Även om jag primärt använder ålder och kön som features, har jag strukturerat koden för att möjliggöra utökad feature engineering i framtida versioner.

Dataseparation: Jag implementerar tränings-/testuppdelning med scikit-learn:

- **Användaren kan justera testproportionen (standardvärde 30%)**
- **Stratifierad uppdelning säkerställer att proportionerna av målklasser bibehålls**
- **En fast slumpfrö (random_state=42) används för reproducerbarhet**

2.3 Modellval och Arkitektur

För att lösa klassificeringsproblemet har jag implementerat tre olika maskininlärningsmodeller:

1. **Gaussian Naive Bayes:**
    - **En probabilistisk klassificerare baserad på Bayes teorem ○ Väl lämpad för mindre dataset och kategoriska målvariabler ○ Snabb träning och prediktion ○ Fungerar även med relativt få samples per klass**
2. **K-Nearest Neighbors (KNN):**
    - **En instansbaserad inlärningsmetod som klassificerar baserat på likheten med träningsexempel**
    - **Särskilt effektiv när liknande demografiska egenskaper förväntas resultera i liknande musiksmak**
    - **Implementerad med n_neighbors=3 för att hitta en balans mellan övergeneralisering och överanpassning**
3. **Random Forest:**
    - **En ensemblemetod som kombinerar flera beslutsträd ○ Robust mot outliers och överanpassning ○ Kan hantera icke-linjära relationer mellan variabler**
    - **Implementerad med n_estimators=100 för tillräcklig variation i ensemblen Dessa modeller utgör tillsammans en mångsidig approach för klassificering:**

- **Naive Bayes representerar en enkel probabilistisk metod**
- **KNN representerar en instansbaserad metod**
- **Random Forest representerar en ensemblemetod**

Användaren kan välja modell baserad på datats egenskaper och önskad balans mellan tolkningsbarhet och precision.

2.4 Träningsprocess

I min applikationslogik implementerar jag en strukturerad träningsprocess med följande steg:

1. **Datauppdelning:**
    - **Data delas upp i tränings- och testset (justerbara proportioner)**
    - **Stratifiering används för att bibehålla klassfördelningen**
2. **Feature-skalning:**
    - **StandardScaler används för att normalisera ålder**
    - **Skalningsparametrar sparas för användning vid prediktion**
3. **Modellträning:**
    - **Den valda modellen tränas på träningsdata**
    - **För Random Forest tränas 100 beslutsträd**
    - **För KNN används 3 grannar**
    - **För Naive Bayes används standardinställningar**
4. **Utvärdering:**
    - **Modellen utvärderas på testdata**
    - **Beräkning av precision, recall, F1-score och konfusionsmatris**
    - **För Random Forest extraheras feature-importance för tolkbarhet**
5. **Modellpersistens:**
    - **Möjlighet att spara tränad modell och skalningsparametrar till fil**
    - **Användaren kan exportera modellen för senare användning eller distribution**

3. Teknisk Implementation

3.1 Utvecklingsmiljö

Jag implementerade projektet med Python och ett antal nyckelbibliotek för datavetenskap och maskininlärning:

Kärnbibliotek:

- **pandas: För datamanipulation och -analys**
- **numpy: För effektiva numeriska beräkningar**
- **matplotlib och seaborn: För datavisualisering**
- **scikit-learn: För maskininlärningsmodeller och utvärderingsverktyg GUI-bibliotek:**
- **tkinter: För grafiskt användargränssnitt**
- **matplotlib.backends.backend_tkagg: För integrering av matplotlib-visualiseringar i tkinter**

Övriga bibliotek:

- **os: För filsystemsoperationer**
- **pickle: För modellpersistens (sparande/laddning)**

Jag strukturerade applikationen som en objektorienterad implementation med MusicPreferencesApp-klassen som central komponent. Denna klass ansvarar för:

- **Initialisering av användargränssnittet**
- **Datainläsning och -bearbetning**
- **Modellinställning och -träning**
- **Visualisering och resultatpresentation**

3.2 Arkitektur och Kodstruktur

Jag valde en modulär arkitektur med tydlig separation av ansvar:

Klassbaserad design: Min huvudklass MusicPreferencesApp innehåller all funktionalitet organiserad i tydliga metoder:

- **init: Initialiserar applikationen och grundläggande komponenter**
- **setup_*_tab: Konfigurerar de olika flikarna i gränssnittet**
- **load_data, use_default_data: Hanterar datainläsning**
- **display_data_info, create_visualizations: Analyserar och visualiserar data**
- **check_and_handle_data_issues: Identifierar och hanterar datakvalitetsproblem**
- **train_model, update_prediction_info: Modellträning och resultatuppdatering**
- **predict_genre: Genomför prediktioner med tränad modell**
- **save_model: Sparar tränad modell till fil**

Tab-baserat gränssnitt: Jag organiserade applikationen i fyra flikar som representerar olika steg i ML-arbetsflödet:

1. **Data Analysis: För datautforskning och visualisering**
2. **Model Training: För val av modell och träningsparametrar**
3. **Prediction: För att göra prediktioner med tränad modell**
4. **About: Information om applikationen och användningsguide**

Denna struktur följer ett logiskt flöde från datainläsning och -analys, genom modellträning till prediktion, vilket speglar det typiska arbetsflödet inom maskininlärning.

3.3 Datautforskning och Visualisering

En central del av mitt projekt är utforskande dataanalys, implementerad genom flera visualiseringar: Distributionstabeller:

- **Sammanfattning av demografisk data (ålder, kön)**
- **Genrefördelning i datasetet Visualiseringskomponenter:**

1. **Åldersfördelning: Histogram som visar åldersfördelningen i datasetet**
2. **Genrefördelning: Stapeldiagram som visar antal individer per musikgenre**
3. **Ålder per genre: Boxplot som visar åldersfördelning inom varje genre**
4. **Genre per kön: Stapeldiagram som visar genrepreferenser uppdelat efter kön**

Dessa visualiseringar genereras dynamiskt baserat på inläst data och uppdateras när användaren importerar ny data eller hanterar datakvalitetsproblem.

3.4 Modellträning och Utvärdering

Träningsprocessen har jag implementerat med hjälp av scikit-learn och den omfattar:

Modellval:

- **Användaren väljer en av de tre implementerade modellerna via dropdown-menyn**
- **Testproportionen kan justeras med ett reglage Träningsprocedur:**

 # Prepare data

X = self.df[['age', 'gender']] y =

self.df['genre'] # Scale features self.scaler

= StandardScaler() X_scaled =

self.scaler.fit_transform(X)

 # Split data

test_size = self.test_size_var.get()

X_train, X_test, y_train, y_test = train_test_split(

X_scaled, y, test_size=test_size, random_state=42)

 # Select and train model model_name =

self.model_var.get() if model_name ==

'Gaussian Naive Bayes':

self.model = GaussianNB() elif model_name

== 'K-Nearest Neighbors':

self.model = KNeighborsClassifier(n_neighbors=3) else: # Random Forest

self.model = RandomForestClassifier(n_estimators=100, random_state=42)

 # Train model self.model.fit(X_train,

y_train)

Utvärderingsmetrik:

- **Träningsprecision och testprecision**
- **Detaljerad klassificeringsrapport (precision, recall, F1-score)**
- **Konfusionsmatris**
- **Feature-importance för Random Forest**

Resultaten presenteras i ett textfält och kan användas för att bedöma modellens prestanda och identifiera potentiella förbättringsområden.

3.5 Prediktion och Användargränssnitt

Den tränade modellen används för att göra prediktioner via en användarvänlig gränssnittsflik: Prediktionsgränssnitt:

- **Inmatningsfält för ålder**
- **Radioknappar för kön**
- **Knapp för att utföra prediktion**
- **Resultatvisning med predikterad genre och konfidensnivå (när tillgänglig)**

Prediktionslogik: # Get input values age = self.age_var.get() gender = self.gender_var.get()

 # Scale the input features =

self.scaler.transform([[age, gender]])

 # Make prediction prediction =

self.model.predict(features)[0]

 # Get probabilities if available if

hasattr(self.model, "predict_proba"):

proba = self.model.predict_proba(features)[0]

max_proba = max(proba) * 100 confidence_text = f"

(Confidence: {max_proba:.1f}%)" else:

confidence_text = ""

Insikter om prediktionsmönster: Min applikation analyserar även det kompletta datasetet för att ge användaren insikter om generella mönster i musikpreferenser:

- **Genomsnittlig ålder per genre**
- **Popularitet av genrer per kön**
- **Identifierade trender i datasetets demografiska egenskaper**

4. Resultat och Utvärdering

4.1 Modellprestanda

Applikationens utvärderingsmodul beräknar och presenterar flera metriker för att bedöma modellprestanda:

Accuracy: Den övergripande precisionen (procentandel korrekta prediktioner) är den primära metriken, men jag kompletterar den med mer nyanserade mått.

Per-klass-metriker: För varje genre (klass) beräknar jag:

- **Precision (andel korrekta av de som predikterades som denna genre)**
- **Recall (andel av faktiska genren som korrekt identifierades)**
- **F1-score (harmoniskt medelvärde av precision och recall)**

Konfusionsmatris: En visuell representation av korrekta vs. felaktiga klassificeringar, vilket hjälper till att identifiera mönster i missklassificeringar.

Feature importance: För Random Forest-modellen visar jag varje features relativa betydelse, vilket ger insikt i huruvida ålder eller kön har större påverkan på musikpreferens.

4.2 Datahantering och Kvalitetssäkring

Jag har implementerat robust datahantering med flera kvalitetssäkrande mekanismer:

Datakvalitetsanalys: När data laddas genomförs automatiskt en kvalitetskontroll som identifierar:

- **Saknade värden**
- **Potentiella extremvärden (outliers)**
- **Datatypdiskrepanser**

Korrigeringsmekanismer: Vid identifierade dataproblem erbjuder jag användaren flera korrigeringsalternativ:

- **För saknade värden: imputering med medelvärde, median, eller borttagning**
- **För outliers: capping, ersättning, eller borttagning**

Datafördelningsanalys: Visualiseringar hjälper användaren att bedöma:

- **Balansen mellan klasser (genrer)**
- **Demografisk representation i datasetet**
- **Potentiella bias i träningsdata**

4.3 Användarvänlighet

En central aspekt av projektet är den användarvänliga implementationen:

Intuitivt flöde: Flikarna är organiserade i logisk sekvens från datainläsning till prediktion.

Informativ återkoppling:

- **Tydliga felmeddelanden vid problem**
- **Visuell feedback om databehandling**
- **Detaljerade resultatrapporter**

Flexibilitet:

- **Stöd för både default-data och egen data**
- **Möjlighet att justera centrala modellparametrar**
- **Export/import-funktionalitet för tränade modeller**

5. Diskussion

5.1 Algoritmval och Prestanda

De tre modellerna jag implementerade (Gaussian Naive Bayes, KNN, Random Forest) representerar olika maskininlärningsparadigm och erbjuder olika fördelar:

Naive Bayes:

- **Fördel: Fungerar väl med små dataset och är beräkningseffektiv**
- **Nackdel: Antagandet om oberoende features kan vara en förenkling KNN:**
- **Fördel: Intuitiv och icke-parametrisk, hanterar komplexa klassfördelningar**
- **Nackdel: Känslig för feature-skalning och dimensionalitet Random Forest:**
- **Fördel: Robust mot överanpassning, hanterar icke-linjära relationer väl**
- **Nackdel: Mindre tolkningsbar än enklare modeller**

För ett typiskt musikpreferensdataset har jag observerat att Random Forest oftast presterar bäst på grund av dess förmåga att fånga icke-linjära relationerer och hantera både numeriska och kategoriska features effektivt.

5.2 Datarepresentation och Begränsningar

I mitt projekt representerar jag demografisk data på ett förenklat sätt med ålder som kontinuerlig variabel och kön som binär variabel. Detta medför vissa begränsningar:

Representation av kön: Den binära kodningen (0/1) är en förenkling som inte fångar könsspektrumets komplexitet.

Åldersrepresentation: Ålder behandlas som en linjär variabel, men jag har insett att musikpreferenser kan förändras icke-linjärt över livscykeln.

Saknade faktorer: Flera potentiellt viktiga faktorer inkluderas inte:

- **Kulturell bakgrund**
- **Geografisk position**
- **Utbildningsnivå**
- **Tidigare musikerfarenheter**

En mer omfattande modell skulle kunna inkludera dessa faktorer för förbättrad prediktionsförmåga.

5.3 Tekniska Utmaningar och Lösningar

Under utvecklingen av projektet hanterade jag flera tekniska utmaningar:

Outlier-hantering: Jag upptäckte att extremvärden i ålder kan påverka modellerna negativt, särskilt KNN. Lösningen var en implementerad outlier-detektionsmodul med flera korrigeringsalternativ.

Feature-skalning: Olika modeller har olika krav på feature-skalning. Jag valde att använda StandardScaler genomgående för att säkerställa optimal modellprestanda.

Klassbalans: Obalanserade klasser kan leda till partiska modeller. Min applikation visualiserar klassfördelningen så att användaren är medveten om potentiella bias.

GUI-responsivitet: Tkinter kan bli långsamt vid komplexa operationer. För att lösa detta använde jag uppdateringsanrop för att hålla gränssnittet responsivt under beräkningsintensiva operationer.

6. Slutsatser och Framtida Arbete

6.1 Huvudsakliga Slutsatser

Mitt projekt demonstrerar ett fullständigt maskininlärningsarbetsflöde för prediktion av musikpreferenser baserat på demografiska faktorer. Genom implementation av tre olika klassificeringsmodeller och omfattande data-visualiseringsverktyg har jag visat hur: 1. Demografiska faktorer har mätbar korrelation med musikgenrepreferenser

2. **Maskininlärningsmodeller kan göra rimliga prediktioner även med begränsad demografisk data**
3. **Ett intuitivt gränssnitt kan göra maskininlärning tillgängligt för icke-tekniska användare**

6.2 Framtida Utvecklingsmöjligheter

För framtida utveckling av projektet har jag identifierat flera potentiella förbättringsområden:

Utökad datamodell:

- **Inkludera fler demografiska faktorer (utbildning, inkomst, etc.)**
- **Samla in och använda historiska lyssningsdata**
- **Införa personlighetsvariabler som features Avancerade modeller:**
- **Implementera djupinlärningsmodeller för komplexa mönster**
- **Utveckla hybridmodeller som kombinerar innehålls- och demografibaserade prediktioner**
- **Utforska kollaborativ filtrering för rekommendationer Funktionella tillägg:**
- **Integration med musikstreamingtjänster**
- **Realtidsprediktioner baserat på lyssningsbeteende**
- **Interaktiva visualiseringar med direkt feedback Tekniska förbättringar:**
- **Optimera för stora dataset**
- **Implementera parallelliserad modellträning**
- **Utveckla batch-prediktionsmöjligheter för stora användargrupper**

6.3 Avslutande Reflektioner

Detta projekt har gett mig värdefull insikt i hur relativt enkla maskininlärningsmodeller kan kopplas till verkliga användningsfall för att skapa värdefulla insikter och prediktioner. Genom att kombinera grundläggande demografisk data med moderna ML-tekniker har jag sett potential för utveckling av mer sofistikerade system för personanpassning inom musikdomänen.

Den implementerade applikationen fungerar inte bara som ett prediktionsverktyg utan även som ett pedagogiskt verktyg för att förstå sambandet mellan demografi och kulturella preferenser. Den transparenta designen och omfattande visualiseringsmöjligheterna gör maskininlärningens svarta låda något mer genomskinlig och förståelig för slutanvändaren.

Slutligen har projektet gett mig praktisk erfarenhet av att använda Python-programmering och standardbibliotek för datavetenskap för att bygga en komplett lösning från datainsamling till användarvänlig applikation.

7. Källförteckning

1. **Müller, A. C., & Guido, S. (2022). Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media.**
2. **McKinney, W. (2022). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.**
3. **Géron, A. (2023). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.**
4. **Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2021.**
5. **Matplotlib: Visualization with Python. Hunter, J. D. (2007). Computing in Science & Engineering, 9(3), 90-95.**
6. **Seaborn: Statistical Data Visualization. Waskom, M. L. (2021). Journal of Open Source Software, 6(60), 3021.**
7. **Pandas: Powerful Python Data Analysis Toolkit. McKinney, W. (2010). Proceedings of the 9th Python in Science Conference, 51-56.**

Appendix A: Datautforskningsexempel

Följande kod demonstrerar hur jag implementerade datautforskning och visualisering i projektet: def create_visualizations(self):

```
    """Create and display visualizations"""     if
self.df is not None:
        # Clear previous visualizations        for widget
in self.viz_frame.winfo_children():
            widget.destroy()

        # Create figure with subplots
fig = Figure(figsize=(8, 8))

        # Age distribution        ax1 = fig.add_subplot(221)
ax1.hist(self.df['age'], bins=10, edgecolor='black')
ax1.set_title('Age Distribution')        ax1.set_xlabel('Age')
ax1.set_ylabel('Count')

        # Genre distribution        ax2 = fig.add_subplot(222)
genre_counts = self.df['genre'].value_counts()
ax2.bar(genre_counts.index, genre_counts.values)
ax2.set_title('Genre Distribution')        ax2.set_xlabel('Genre')
ax2.set_ylabel('Count')        plt.setp(ax2.get_xticklabels(),
rotation=45, ha='right')

        #   Age   by   genre   boxplot
ax3     =     fig.add_subplot(223)
genre_order                          =
genre_counts.index.tolist()

        sns.boxplot(x='genre', y='age', data=self.df, ax=ax3, order=genre_order)
ax3.set_title('Age Distribution by Genre')        ax3.set_xlabel('Genre')
ax3.set_ylabel('Age')        plt.setp(ax3.get_xticklabels(), rotation=45, ha='right')
```

```
        # Genre by gender        ax4 = fig.add_subplot(224)
gender_genre = pd.crosstab(self.df['gender'], self.df['genre'])
gender_genre.plot(kind='bar', stacked=True, ax=ax4)
ax4.set_title('Genre Preferences by Gender')
ax4.set_xlabel('Gender (0=Female, 1=Male)')
ax4.set_ylabel('Count')        ax4.legend(title='Genre')

        fig.tight_layout()

        # Add the plot to the GUI        canvas =
FigureCanvasTkAgg(fig, master=self.viz_frame)
canvas.draw()        canvas.get_tk_widget().pack(fill=tk.BOTH,
expand=True)
```

Appendix B: Modellträningsexempel

Följande kodsegment visar hur jag implementerade modellträning och utvärdering i projektet:

```
def train_model(self):
    """Train the selected machine learning model"""     if
self.df is None:
        messagebox.showwarning("Warning", "Please load data first!")        return

    try:
        # Clear previous results        self.results_text.config(state=tk.NORMAL)
self.results_text.delete(1.0, tk.END)        self.results_text.insert(tk.END,
"Training model...\n\n")        self.results_text.update()

        # Prepare data
        X = self.df[['age', 'gender']]
y = self.df['genre']
```

```
        # Scale features        self.scaler =
StandardScaler()        X_scaled =
self.scaler.fit_transform(X)

        # Split data
        test_size = self.test_size_var.get()
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=test_size, random_state=42)

        # Select model        model_name =
self.model_var.get()        if model_name ==
'Gaussian Naive Bayes':
            self.model = GaussianNB()        elif
model_name == 'K-Nearest Neighbors':
            self.model = KNeighborsClassifier(n_neighbors=3)        else: # Random Forest
self.model = RandomForestClassifier(n_estimators=100, random_state=42)

        # Train model
        self.model.fit(X_train, y_train)

        # Make predictions        y_pred =
self.model.predict(X_test)

        # Evaluate        acc = accuracy_score(y_test,
y_pred)        report = classification_report(y_test,
y_pred)        cm = confusion_matrix(y_test, y_pred)

        # Display results        self.results_text.insert(tk.END, f"Model:
{model_name}\n")        self.results_text.insert(tk.END, f"Test Size:
{test_size:.2f}\n\n")        self.results_text.insert(tk.END, f"Accuracy:
```

```
{acc:.4f}\n\n")        self.results_text.insert(tk.END, "Classification
Report:\n")        self.results_text.insert(tk.END, f"{report}\n")
self.results_text.insert(tk.END, "Confusion Matrix:\n")
self.results_text.insert(tk.END, f"{cm}\n\n")

      # Feature importance for Random Forest        if
isinstance(self.model, RandomForestClassifier):
self.results_text.insert(tk.END, "Feature Importance:\n")
importance = self.model.feature_importances_            for i, feat in
enumerate(['Age', 'Gender']):
            self.results_text.insert(tk.END, f" {feat}: {importance[i]:.4f}\n")

      self.results_text.config(state=tk.DISABLED)

      # Update prediction info
self.update_prediction_info()
messagebox.showinfo("Success", f"{model_name} trained
successfully!")

   except Exception as e:
      messagebox.showerror("Error", f"Error during training: {str(e)}")
```

Appendix C: Felanalys och Datakvalitetshantering

Nedan visas hur jag implementerade datakvalitetshantering och feldetektering: def check_and_handle_data_issues(self):

```
    """Check and handle missing values and outliers in the dataset"""     if self.df is None:
        return

    # Check for missing values     missing_values = self.df.isnull().sum()     has_missing = missing_values.sum() > 0

    # Check for outliers in age using IQR method
    Q1 = self.df['age'].quantile(0.25)
    Q3 = self.df['age'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

    age_outliers = self.df[(self.df['age'] < lower_bound) | (self.df['age'] > upper_bound)]
has_outliers = len(age_outliers) > 0

    # No issues found     if not has_missing and not has_outliers:
        messagebox.showinfo("Data Check", "No missing values or outliers found in the dataset.")
        return

    # Create a data issues report     report = "Data Issues Report:\n\n"

    if has_missing:
```

```
      report += "Missing Values Found:\n"
for col, count in missing_values.items():
if count > 0:
            report += f" {col}: {count} missing values\n"
report += "\n"


   if has_outliers:
      report += f"Age Outliers Found: {len(age_outliers)} records\n"        report +=
f" Age Range: {self.df['age'].min()} - {self.df['age'].max()}\n"        report += f"
Expected Range: {lower_bound:.2f} - {upper_bound:.2f}\n\n"


   # Show dialog with the report and options     self.show_data_issues_dialog(report,
has_missing, has_outliers, Q1, Q3, IQR)
```

Appendix D: Prediktionsimplementation

Följande kodsegment visar hur jag implementerade prediktionsfunktionen i applikationen: def predict_genre(self):

```
    """Make a prediction based on user input"""     if self.model is None or self.scaler is None:
        messagebox.showwarning("Warning", "Please train a model first!")         return

    try:
        # Get input values         age = self.age_var.get()         gender = self.gender_var.get()

        # Scale the input         features = self.scaler.transform([[age, gender]])

        # Make prediction         prediction = self.model.predict(features)[0]

        # Get probabilities if available         if hasattr(self.model, "predict_proba"):
            proba = self.model.predict_proba(features)[0]
max_proba = max(proba) * 100             confidence_text = f" (Confidence: {max_proba:.1f}%)"         else:
            confidence_text = ""

        # Update result label         gender_text = "Male" if gender == 1 else "Female"         result_text = f"Predicted Genre: {prediction}{confidence_text}"
self.prediction_result.config(text=result_text)
```

```
except Exception as e:
    messagebox.showerror("Error", f"Error during prediction: {str(e)}")
```

Appendix E: OOP-implementation

Min objektorienterade design implementerar följande centrala principer inom OOP:

Inkapsling: Klassen MusicPreferencesApp inkapslar all data och funktionalitet för applikationen. Användarkod behöver endast interagera med instantiering av klassen och anrop till mainloop():

```
def main():
    root = tk.Tk()
    app = MusicPreferencesApp(root)    root.mainloop()


if __name__ == "__main__":
    main()
```

Privata och Publika Metoder: Jag organiserade metoder baserat på deras avsedda användning. Publika metoder som train_model() och predict_genre() exponeras för externa anrop, medan hjälpfunktioner som display_data_info() är avsedda för intern användning.

Datapersistens: Jag implementerade både temporär och långvarig datapersistens:

```
def save_model(self):
    """Save the trained model to a file"""
if self.model is None:
        messagebox.showwarning("Warning", "Please train a model first!")        return
    try:
        file_path = filedialog.asksaveasfilename(
title="Save Model",            defaultextension=".pkl",
filetypes=[("Pickle files", "*.pkl"), ("All files", "*.*")]        )
        if file_path:            with
open(file_path, 'wb') as f:
            # Save both the model and the scaler
pickle.dump((self.model, self.scaler), f)
messagebox.showinfo("Success", "Model saved successfully!")

    except Exception as e:
        messagebox.showerror("Error", f"Error saving model: {str(e)}")
```

Appendix F: Koppling till Maskininlärningskoncept

Mitt projekt demonstrerar flera centrala koncept inom maskininlärning genom praktisk implementation:

Supervised Learning: Jag implementerar supervised learning där modeller tränas på labelad data (ålder och kön som features, musikgenre som label).

Feature Scaling: Jag visar vikten av feature scaling för att säkerställa att features med olika skalor

behandlas rättvist av modellen: self.scaler = StandardScaler() X_scaled =

self.scaler.fit_transform(X)

Cross-Validation: Tränings-/testuppdelning används för att utvärdera modellen på osedd data:

X_train, X_test, y_train, y_test = train_test_split(

X_scaled, y, test_size=test_size, random_state=42)

Ensemble Learning: Random Forest-modellen demonstrerar ensemble learning där prediktioner från flera modeller kombineras:

self.model = RandomForestClassifier(n_estimators=100, random_state=42)

Model Evaluation: Jag implementerar omfattande modellutvärdering:

acc = accuracy_score(y_test, y_pred)

report = classification_report(y_test, y_pred) cm =

confusion_matrix(y_test, y_pred)

Appendix G: Sammanfattning av Programflöde

Detta diagram illustrerar det övergripande programflödet i min applikation:

1. **Initialisering**
   - **Skapa GUI-komponenter**
   - **Konfigurera flikar och kontroller**
2. **Datainläsning** ○ **Ladda data från CSV eller använd default-data** ○ **Validera och förbehandla data**
3. **Datavisualisering** ○ **Generera visualiseringar** ○ **Visa statistisk sammanfattning**
4. **Datakvalitetshantering** ○ **Identifiera och hantera saknade värden** ○ **Identifiera och hantera outliers**
5. **Modellträning** ○ **Standardisera features** ○ **Dela data i tränings- och testset** ○ **Träna vald modell**
6. **Modellutvärdering** ○ **Beräkna precision, recall, F1-score** ○ **Visa konfusionsmatris** ○ **Visa feature importance**
7. **Prediktion** ○ **Ta ålder och kön som input** ○ **Standardisera input-features** ○ **Generera prediktion med konfidensnivå**
8. **Modellpersistens**
   - **Spara tränad modell och scaler** ○ **Möjliggör återanvändning**