Do data science (ideally deep learning but simpler one is fine) using chatgpt by picking a popular data set in kaggle website, upload and do various modules of data science - like my example . Export your chat transcript and submit it. publish a medium article by selecting sections of the output lets use titanic data set

ChatGPT said:

Thought for 36s

Analyzed

Always show details

```
# Titanic Kaggle end-to-end: EDA → Feature Engineering → Modeling → Medium-ready report
#
# This notebook-style script will:
# 1) Try to load Kaggle's titanic `train.csv` if you upload it (use the file uploader here).
#    If not found, it will create a small mock dataset so you can see the full workflow immediately.
# 2) Run light EDA and feature engineering.
# 3) Train a few models (LogReg, RandomForest, GradientBoosting) with cross-validation.
# 4) Evaluate on a held-out test split with metrics and plots (matplotlib only, as required).
# 5) Save artifacts (model + figures) and generate a Medium-article draft markdown you can publish.
#
# After this runs, you'll get links to download the artifacts (model + report).
# If you later upload the real Kaggle `train.csv`, just re-run this cell to reproduce with real data.

import os
import re
import textwrap
from pathlib import Path

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_validate
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, f1_score, roc_auc_score, classification_report,
    confusion_matrix, RocCurveDisplay
)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.impute import SimpleImputer
import joblib

# ---------- 0) Load data (or build a mock fallback) ----------
DATA_DIR = Path("/mnt/data")
TRAIN_PATH = DATA_DIR / "train.csv"

def load_titanic_or_mock():
    if TRAIN_PATH.exists():
        df = pd.read_csv(TRAIN_PATH)
        source = "kaggle"
        return df, source
    # Minimal mock that mirrors Kaggle columns for demo
    rng = np.random.default_rng(42)
    n = 200
    df = pd.DataFrame({
        "PassengerId": np.arange(1, n+1),
        "Survived": rng.integers(0, 2, size=n),
        "Pclass": rng.integers(1, 4, size=n),
        "Name": [f"Surname{i}, Mr. Mock" for i in range(1, n+1)],
        "Sex": rng.choice(["male", "female"], size=n, replace=True),
        "Age": np.clip(rng.normal(30, 14, size=n), 0.42, 80),
        "SibSp": rng.integers(0, 4, size=n),
        "Parch": rng.integers(0, 3, size=n),
        "Ticket": rng.choice(["A/5 21171", "PC 17599", "STON/O2.
3101282"], size=n),
```

```
        "Fare": np.abs(rng.normal(32, 49, size=n)),
        "Cabin": rng.choice([np.nan, "C85", "C123", "E46"], size=n,
p=[0.7, 0.1, 0.1, 0.1]),
        "Embarked": rng.choice(["S", "C", "Q"], size=n, p=[0.7, 0.2,
0.1])
    })
    # Introduce some missingness
    mask = rng.choice([True, False], size=n, p=[0.1, 0.9])
    df.loc[mask, "Age"] = np.nan
    df.loc[rng.choice(df.index, size=10, replace=False), "Embarked"] =
np.nan
    source = "mock"
    return df, source

df_raw, data_source = load_titanic_or_mock()

# Show a peek to the user (interactive table)
from caas_jupyter_tools import display_dataframe_to_user
display_dataframe_to_user("Titanic sample (first 100 rows)",
df_raw.head(100))

# ---------- 1) EDA (lightweight) ----------
# Summary stats
eda_summary = df_raw.describe(include="all").transpose()

# Target balance (if Survived present)
target_balance = None
if "Survived" in df_raw.columns:
    target_balance = df_raw["Survived"].value_counts(normalize=True)

# ---------- 2) Feature engineering ----------
df = df_raw.copy()

# Title extraction from Name
def extract_title(name):
    if pd.isna(name):
        return np.nan
    m = re.search(r",\s*([^\.]+)\.", name)
```

```
    return m.group(1).strip() if m else np.nan

if "Name" in df.columns:
    df["Title"] = df["Name"].apply(extract_title)
else:
    df["Title"] = np.nan

# Family size and isolation
df["FamilySize"] = df.get("SibSp", 0) + df.get("Parch", 0) + 1
df["IsAlone"] = (df["FamilySize"] == 1).astype(int)

# Cabin known flag
df["HasCabin"] = (~df["Cabin"].isna()).astype(int)

# Ticket group size proxy (first token of ticket)
def ticket_prefix(ticket):
    if pd.isna(ticket):
        return np.nan
    return str(ticket).split()[0]

df["TicketPrefix"] = df.get("Ticket", np.nan).apply(ticket_prefix)

# Select features
target_col = "Survived" if "Survived" in df.columns else None
features = [
    "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked",
    "Title", "FamilySize", "IsAlone", "HasCabin", "TicketPrefix"
]

X = df[features].copy()
y = df[target_col].copy() if target_col else None

# Train/Valid split
if y is not None:
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=42
    )
else:
```

```
    # Shouldn't happen for Kaggle train.csv, but keep safe defaults
    X_train, X_test, y_train, y_test = X.copy(), X.copy(), None, None

# Preprocess: impute + one-hot
numeric_features = ["Age", "SibSp", "Parch", "Fare", "FamilySize",
"IsAlone", "HasCabin"]
categorical_features = ["Pclass", "Sex", "Embarked", "Title",
"TicketPrefix"]

numeric_transformer = SimpleImputer(strategy="median")
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore", sparse=False))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)

# Models
models = {
    "LogisticRegression": LogisticRegression(max_iter=200, n_jobs=None
if hasattr(LogisticRegression, 'n_jobs') else None),
    "RandomForest": RandomForestClassifier(n_estimators=300,
random_state=42),
    "GradientBoosting": GradientBoostingClassifier(random_state=42)
}

# Build pipelines
pipelines = {
    name: Pipeline(steps=[("preprocess", preprocess), ("model", mdl)])
    for name, mdl in models.items()
}

# Cross-validate
```

```
cv_results = {}
if y is not None:
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    scoring = {"accuracy": "accuracy", "f1": "f1", "roc_auc":
"roc_auc"}
    for name, pipe in pipelines.items():
        scores = cross_validate(pipe, X_train, y_train, cv=cv,
scoring=scoring, n_jobs=None, return_train_score=False)
        cv_results[name] = {
            "accuracy_mean": np.mean(scores["test_accuracy"]),
            "f1_mean": np.mean(scores["test_f1"]),
            "roc_auc_mean": np.mean(scores["test_roc_auc"]),
        }

cv_df = pd.DataFrame(cv_results).T.sort_values(by="roc_auc_mean",
ascending=False) if cv_results else pd.DataFrame()

# Fit best by roc_auc
best_name = cv_df.index[0] if not cv_df.empty else "RandomForest"
best_pipe = pipelines[best_name]
if y is not None:
    best_pipe.fit(X_train, y_train)

# Evaluate on hold-out
test_metrics = {}
conf_mat = None
roc_disp = None

FIG_DIR = DATA_DIR / "figures"
FIG_DIR.mkdir(parents=True, exist_ok=True)

if y is not None:
    y_prob = best_pipe.predict_proba(X_test)[:, 1]
    y_pred = (y_prob >= 0.5).astype(int)

    test_metrics["accuracy"] = float(accuracy_score(y_test, y_pred))
    test_metrics["f1"] = float(f1_score(y_test, y_pred))
    try:
```

```
        test_metrics["roc_auc"] = float(roc_auc_score(y_test, y_prob))
    except Exception:
        test_metrics["roc_auc"] = None

    # Confusion matrix plot
    conf_mat = confusion_matrix(y_test, y_pred)
    plt.figure()
    plt.imshow(conf_mat, interpolation="nearest")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    for (i, j), val in np.ndenumerate(conf_mat):
        plt.text(j, i, int(val), ha="center", va="center")
    cm_path = FIG_DIR / "confusion_matrix.png"
    plt.savefig(cm_path, bbox_inches="tight")
    plt.close()

    # ROC curve
    try:
        RocCurveDisplay.from_predictions(y_test, y_prob)
        roc_path = FIG_DIR / "roc_curve.png"
        plt.title("ROC Curve")
        plt.savefig(roc_path, bbox_inches="tight")
        plt.close()
    except Exception:
        roc_path = None

# Feature importances / coefficients (try to extract for best model)
feature_summary = None
try:
    # Fit a fresh pipeline to get named columns after preprocess
    best_pipe.fit(X_train, y_train)
    # Get names after one-hot
    ohe =
best_pipe.named_steps["preprocess"].transformers_[1][1].named_steps["o
he"]
    ohe_cols = list(ohe.get_feature_names_out(categorical_features))
    final_columns = numeric_features + ohe_cols
```

```
    if best_name == "RandomForest":
        importances =
best_pipe.named_steps["model"].feature_importances_
        feature_summary = pd.DataFrame({"feature": final_columns,
"importance": importances}).sort_values("importance", ascending=False)
    elif best_name == "LogisticRegression":
        coefs = best_pipe.named_steps["model"].coef_[0]
        feature_summary = pd.DataFrame({"feature": final_columns,
"coef": coefs}).sort_values("coef", key=np.abs, ascending=False)
    elif best_name == "GradientBoosting":
        importances =
best_pipe.named_steps["model"].feature_importances_
        feature_summary = pd.DataFrame({"feature": final_columns,
"importance": importances}).sort_values("importance", ascending=False)
except Exception as e:
    feature_summary = pd.DataFrame({"feature": [], "importance": []})

# Save model
MODEL_PATH = DATA_DIR / "titanic_best_model.joblib"
if y is not None:
    joblib.dump({"pipeline": best_pipe, "best_name": best_name,
"cv_table": cv_df, "test_metrics": test_metrics}, MODEL_PATH)

# ---------- 3) Medium-article draft ----------
def md_section(title, body):
    return f"## {title}\n\n{body}\n\n"

intro = textwrap.dedent(f"""
    # Surviving the Titanic with Machine Learning: A Gentle Deep Dive

    *Dataset:* Kaggle Titanic (train.csv). In this post, we walk
through an end-to-end workflow—
    from quick EDA and feature engineering to model selection and
evaluation—using plain Python
    and scikit-learn. The goal is to predict `Survived` from passenger
attributes.
```

```
    *Repro tip:* You can reproduce everything by placing Kaggle's
`train.csv` next to this notebook
    and running the single script cell.
""").strip()

data_section = textwrap.dedent(f"""
    **Columns (typical):** `Pclass, Sex, Age, SibSp, Parch, Fare,
Embarked, Cabin, Ticket, Name, Survived`.

    **Target:** `Survived` (0 = did not survive, 1 = survived).

    **Class balance:** {target_balance.to_dict() if target_balance is
not None else "n/a (mock)"}.
""").strip()

eda_section = textwrap.dedent("""
    We begin with a quick sanity check via summary statistics and a
peek at the first rows.
    Age and Cabin are often missing; Fare has a heavy tail; Embarked
may have a few NaNs.
    These observations drive our imputation choices below.
""").strip()

fe_section = textwrap.dedent("""
    **Feature Engineering**

    - **Title** extracted from `Name` (e.g., Mr, Mrs, Miss, Master).
    - **FamilySize** = `SibSp + Parch + 1` and **IsAlone** indicator.
    - **HasCabin** flag from `Cabin` non-missingness.
    - **TicketPrefix** as a rough proxy for fare/class grouping.

    **Preprocessing**

    - Numeric: median impute for `Age, SibSp, Parch, Fare, FamilySize,
IsAlone, HasCabin`.
    - Categorical: most-frequent impute + one-hot encode for `Pclass,
Sex, Embarked, Title, TicketPrefix`.
""").strip()
```

```
modeling_section = textwrap.dedent(f"""
    **Models compared (5-fold Stratified CV on train split):**

    {cv_df.round(4).to_markdown(index=True) if not cv_df.empty else
"CV results will appear after loading real Kaggle data."}

    We select the model with the highest ROC-AUC on average. In
practice, Logistic Regression
    offers interpretability; Random Forest and Gradient Boosting often
capture non-linearities.
""").strip()

results_section = textwrap.dedent(f"""
    **Hold-out Test Performance (20% split):**

    {pd.Series(test_metrics).round(4).to_frame("score").to_markdown()
if test_metrics else "Metrics available after loading real Kaggle
data."}

    We also inspect the confusion matrix and ROC curve to understand
trade-offs.
""").strip()

fi_section = textwrap.dedent(f"""
    **What drives predictions?**

    {feature_summary.head(15).to_markdown(index=False) if
feature_summary is not None and not feature_summary.empty else
"Feature importances/coefficients will populate after training."}
""").strip()

next_steps = textwrap.dedent("""
    **Next Steps**
    - Calibrate probabilities and tune thresholds for
application-specific costs.
    - Try more advanced models (e.g., CatBoost, XGBoost) and
hyperparameter search.
```

```
    - Engineer richer titles (map rare titles), age-by-class
interactions, deck extraction from Cabin.
    - Perform k-fold cross-validation on the full dataset and submit
to Kaggle for leaderboard feedback.
""").strip()

medium_md = "\n\n".join([intro, md_section("Dataset & Setup",
data_section),
                         md_section("EDA Highlights", eda_section),
                         md_section("Feature Engineering &
Preprocessing", fe_section),
                         md_section("Modeling & Cross-Validation",
modeling_section),
                         md_section("Results", results_section),
                         md_section("Feature Importance",
fi_section),
                         md_section("Next Steps", next_steps)])

MD_PATH = DATA_DIR / "titanic_medium_article.md"
with open(MD_PATH, "w", encoding="utf-8") as f:
    f.write(medium_md)

# ---------- 4) Show key tables to the user ----------
display_dataframe_to_user("EDA summary (describe)",
eda_summary.reset_index())
if not cv_df.empty:
    display_dataframe_to_user("Cross-validation results",
cv_df.reset_index().rename(columns={"index": "model"}))
if feature_summary is not None and not feature_summary.empty:
    display_dataframe_to_user("Top features",
feature_summary.head(30))

# ---------- 5) Save a concise README with instructions ----------
README = f"""
# Titanic ML Pipeline (ChatGPT-run)

## How to Reproduce with Kaggle Data
1. Download `train.csv` from Kaggle Titanic.
```

```
2. Upload `train.csv` here.
3. Re-run the big Python cell.
4. Your artifacts will be saved to `/mnt/data`:
   - `titanic_best_model.joblib` (fitted pipeline)
   - `figures/confusion_matrix.png`
   - `figures/roc_curve.png`
   - `titanic_medium_article.md` (ready-to-publish draft for Medium)

## Notes
- If `train.csv` is not uploaded, a mock dataset is used so you can preview the full workflow.
- Plots are generated using matplotlib only.
"""
README_PATH = DATA_DIR / "README_TITANIC_CHATGPT.txt"
with open(README_PATH, "w", encoding="utf-8") as f:
    f.write(README)

print("Data source used:", data_source)
print("Artifacts saved:")
print(" - Model:", MODEL_PATH if MODEL_PATH.exists() else "Model will save after real Kaggle data.")
print(" - Medium draft:", MD_PATH)
print(" - Figures:", FIG_DIR / "confusion_matrix.png", "|", FIG_DIR / "roc_curve.png")
```

STDOUT/STDERR
/home/sandbox/.local/lib/python3.11/site-packages/pandas/core/internals/blocks.py:2323: RuntimeWarning: invalid value encountered in cast
  values = values.astype(str)
/home/sandbox/.local/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/home/sandbox/.local/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Data source used: mock
Artifacts saved:
 - Model: /mnt/data/titanic_best_model.joblib
 - Medium draft: /mnt/data/titanic_medium_article.md
 - Figures: /mnt/data/figures/confusion_matrix.png | /mnt/data/figures/roc_curve.png

Titanic sample (first 100 rows)

PassengerId	Survived	Pclass	Name
1	0	2	Surname1, Mr. Mock
2	1	3	Surname2, Mr. Mock
3	1	2	Surname3, Mr. Mock
4	0	3	Surname4, Mr. Mock
5	0	2	Surname5, Mr. Mock
6	1	1	Surname6, Mr. Mock
7	0	3	Surname7, Mr. Mock

EDA summary (describe)

index	count	unique	top
PassengerId	200.0		
Survived	200.0		
Pclass	200.0		
Name	200	200	Surname1, Mr. Mock
Sex	200	2	female
Age	187.0		

SibSp	200.0

Cross-validation results

model	accuracy_mean	f1_mean	roc_auc_mean
RandomForest	0.49375	0.4779667644183774	0.4843014705882352 5
GradientBoosting	0.4875	0.4780384307519032	0.4768841911764706
LogisticRegression	0.48125	0.4867138225758915	0.4737775735294118

Top features

	feature	importance
3	Fare	0.23276031604315608
0	Age	0.2159291352926522
4	FamilySize	0.08465886302993524
1	SibSp	0.07938679560226947
2	Parch	0.05222971372225691

Awesome—let's build the Titanic project end-to-end right here, then package it for your submission + Medium post.

I just ran a full pipeline that:

- Loads Kaggle's `train.csv` **if you upload it**; if not, it uses a small mock dataset so the whole workflow still runs.
- Does quick EDA, feature engineering (Title, FamilySize, IsAlone, HasCabin, TicketPrefix), preprocessing (impute + one-hot), model comparison (LogReg, RandomForest, GradientBoosting) with CV, test metrics, confusion matrix, and ROC curve.
- Saves a **publish-ready Medium draft** and a fitted model.

What you should do now

1. If you have Kaggle's Titanic data, upload `train.csv` here and **re-run** the big Python cell above.
2. Grab the outputs for your submission + article:

- **Medium draft (Markdown):**
  Download `titanic_medium_article.md`
- **Model artifact:**
  Download `titanic_best_model.joblib`
- **Figures for the article:**
  - Confusion Matrix
  - ROC Curve
- **Repro notes / instructions:**
  Download `README_TITANIC_CHATGPT.txt`

You'll also see interactive tables in the workspace (EDA summary, cross-validation results, and top features). Once you upload the real Kaggle `train.csv` and re-run, those tables and metrics will reflect the true dataset.

How to submit your "ChatGPT transcript"

Most classes just want the exported chat. In ChatGPT, use the **Share** / **Export** option to save this conversation as a PDF link or file. Pair that with the artifacts above.

How to publish the Medium article

- Open `titanic_medium_article.md` in any editor, copy the contents into Medium's editor, and insert the two images (confusion matrix + ROC) where you want them.
- Optional: tweak the intro or results after you re-run with the real Kaggle data so your numbers match.

If you want, I can also generate a Kaggle-submission `gender_submission.csv` from the fitted model (requires the real `test.csv`). Upload `test.csv` and I'll add that step and save the file for you.

ChatGPT said:
Analyzed
Always show details

```
# User uploaded real Kaggle Titanic train.csv and test.csv.
# Let's re-run the full workflow with these real files, then produce a
Kaggle submission file (`gender_submission.csv` style).

import pandas as pd
import numpy as np
import re
```

```
import matplotlib.pyplot as plt
import joblib

from pathlib import Path
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_validate
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.impute import SimpleImputer

DATA_DIR = Path("/mnt/data")
TRAIN_PATH = DATA_DIR / "train.csv"
TEST_PATH = DATA_DIR / "test.csv"

train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)

# Feature engineering function
def preprocess_titanic(df):
    df = df.copy()
    # Title
    def extract_title(name):
        if pd.isna(name):
            return np.nan
        m = re.search(r",\s*([^\.]+)\.", name)
        return m.group(1).strip() if m else np.nan
    df["Title"] = df["Name"].apply(extract_title)
    # Family size + alone
    df["FamilySize"] = df["SibSp"] + df["Parch"] + 1
    df["IsAlone"] = (df["FamilySize"] == 1).astype(int)
    # Cabin flag
    df["HasCabin"] = (~df["Cabin"].isna()).astype(int)
    # Ticket prefix
```

```
    def ticket_prefix(ticket):
        if pd.isna(ticket):
            return np.nan
        return str(ticket).split()[0]
    df["TicketPrefix"] = df["Ticket"].apply(ticket_prefix)
    return df

train_df = preprocess_titanic(train_df)
test_df = preprocess_titanic(test_df)

target = "Survived"
features = [
    "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked",
    "Title", "FamilySize", "IsAlone", "HasCabin", "TicketPrefix"
]

X = train_df[features]
y = train_df[target]

# Train/valid split
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

numeric_features = ["Age", "SibSp", "Parch", "Fare", "FamilySize",
"IsAlone", "HasCabin"]
categorical_features = ["Pclass", "Sex", "Embarked", "Title",
"TicketPrefix"]

numeric_transformer = SimpleImputer(strategy="median")
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore", sparse=False))
])
preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
```

```
)

# Models
models = {
    "LogisticRegression": LogisticRegression(max_iter=500),
    "RandomForest": RandomForestClassifier(n_estimators=300,
random_state=42),
    "GradientBoosting": GradientBoostingClassifier(random_state=42)
}

pipelines = {name: Pipeline([("preprocess", preprocess), ("model",
mdl)]) for name, mdl in models.items()}

# Cross-validate to choose best
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = {"accuracy": "accuracy", "f1": "f1", "roc_auc": "roc_auc"}

cv_results = {}
for name, pipe in pipelines.items():
    scores = cross_validate(pipe, X_train, y_train, cv=cv,
scoring=scoring)
    cv_results[name] = {
        "accuracy_mean": np.mean(scores["test_accuracy"]),
        "f1_mean": np.mean(scores["test_f1"]),
        "roc_auc_mean": np.mean(scores["test_roc_auc"]),
    }

cv_df = pd.DataFrame(cv_results).T.sort_values("roc_auc_mean",
ascending=False)

best_name = cv_df.index[0]
best_pipe = pipelines[best_name]
best_pipe.fit(X, y)

# Evaluate on validation
y_val_prob = best_pipe.predict_proba(X_valid)[:, 1]
y_val_pred = (y_val_prob >= 0.5).astype(int)
```

```
val_metrics = {
    "accuracy": accuracy_score(y_valid, y_val_pred),
    "f1": f1_score(y_valid, y_val_pred),
    "roc_auc": roc_auc_score(y_valid, y_val_prob)
}

# Generate submission for Kaggle
X_test = test_df[features]
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": best_pipe.predict(X_test)
})

SUB_PATH = DATA_DIR / "submission.csv"
submission.to_csv(SUB_PATH, index=False)

cv_df, val_metrics, SUB_PATH
```

STDOUT/STDERR
/home/sandbox/.local/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:444:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(

Analyzed
Always show details

```
# Retry with more efficient approach: use only RandomForest (fast &
solid) instead of all three, to avoid timeouts.

train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)

def preprocess_titanic(df):
    df = df.copy()
    def extract_title(name):
```

```
        if pd.isna(name):
            return np.nan
        m = re.search(r",\s*([^\.]+)\.", name)
        return m.group(1).strip() if m else np.nan
    df["Title"] = df["Name"].apply(extract_title)
    df["FamilySize"] = df["SibSp"] + df["Parch"] + 1
    df["IsAlone"] = (df["FamilySize"] == 1).astype(int)
    df["HasCabin"] = (~df["Cabin"].isna()).astype(int)
    def ticket_prefix(ticket):
        if pd.isna(ticket):
            return np.nan
        return str(ticket).split()[0]
    df["TicketPrefix"] = df["Ticket"].apply(ticket_prefix)
    return df

train_df = preprocess_titanic(train_df)
test_df = preprocess_titanic(test_df)

target = "Survived"
features = [
    "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked",
    "Title", "FamilySize", "IsAlone", "HasCabin", "TicketPrefix"
]

X = train_df[features]
y = train_df[target]

from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

numeric_features = ["Age", "SibSp", "Parch", "Fare", "FamilySize",
"IsAlone", "HasCabin"]
categorical_features = ["Pclass", "Sex", "Embarked", "Title",
"TicketPrefix"]

numeric_transformer = SimpleImputer(strategy="median")
categorical_transformer = Pipeline([
```

```
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore", sparse=False))
])
preprocess = ColumnTransformer([
    ("num", numeric_transformer, numeric_features),
    ("cat", categorical_transformer, categorical_features)
])

from sklearn.ensemble import RandomForestClassifier
rf_pipe = Pipeline([("preprocess", preprocess), ("model",
RandomForestClassifier(n_estimators=300, random_state=42))])
rf_pipe.fit(X_train, y_train)

# Validation
y_val_prob = rf_pipe.predict_proba(X_valid)[:, 1]
y_val_pred = (y_val_prob >= 0.5).astype(int)

val_metrics = {
    "accuracy": accuracy_score(y_valid, y_val_pred),
    "f1": f1_score(y_valid, y_val_pred),
    "roc_auc": roc_auc_score(y_valid, y_val_prob)
}

# Submission
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": rf_pipe.predict(test_df[features])
})
SUB_PATH = DATA_DIR / "submission.csv"
submission.to_csv(SUB_PATH, index=False)

val_metrics, SUB_PATH
```

Looks like training with 300 trees is still too heavy for the sandbox. We'll switch to a **lighter RandomForest** (say 100 trees) so it finishes quickly but still gives you a valid Kaggle `submission.csv`.

Do you want me to re-run with:

- **RandomForest (n_estimators=100)** → fast + decent accuracy,
- Or a **Logistic Regression** baseline → very fast, interpretable, but possibly lower Kaggle score?

👉 Which option do you prefer for generating your `submission.csv` file?

You said:
random forest

ChatGPT said:
Analyzed
Always show details

```
# Use RandomForest with fewer estimators for speed (n_estimators=100).
from sklearn.ensemble import RandomForestClassifier

rf_pipe = Pipeline([
    ("preprocess", preprocess),
    ("model", RandomForestClassifier(n_estimators=100,
random_state=42))
])

rf_pipe.fit(X_train, y_train)

# Validation evaluation
y_val_prob = rf_pipe.predict_proba(X_valid)[:, 1]
y_val_pred = (y_val_prob >= 0.5).astype(int)

val_metrics = {
    "accuracy": accuracy_score(y_valid, y_val_pred),
    "f1": f1_score(y_valid, y_val_pred),
    "roc_auc": roc_auc_score(y_valid, y_val_prob)
}

# Kaggle submission file
submission = pd.DataFrame({
    "PassengerId": test_df["PassengerId"],
    "Survived": rf_pipe.predict(test_df[features])
})
SUB_PATH = DATA_DIR / "submission.csv"
submission.to_csv(SUB_PATH, index=False)
```

```
val_metrics, SUB_PATH
```

Result
({'accuracy': 0.8100558659217877,
  'f1': 0.7424242424242424,
  'roc_auc': 0.8492753623188406},
 PosixPath('/mnt/data/submission.csv'))

✅ Done! I trained a **RandomForest (100 trees)** and produced your Kaggle-ready file.

Validation metrics (20% hold-out split)

- **Accuracy:** 0.81
- **F1:** 0.74
- **ROC AUC:** 0.85

Kaggle submission file

You can download and submit it directly:
📂 submission.csv

Would you like me to also update your **Medium draft** (`titanic_medium_article.md`) with these *real results* (validation metrics + RandomForest choice), so your article matches the dataset you just used?

You said:

yes

ChatGPT said:

Analyzed
Always show details

```
# Update Medium draft with real Kaggle Titanic results (RandomForest
with n_estimators=100).
import textwrap

# Validation metrics dictionary is available as val_metrics
metrics_table =
pd.Series(val_metrics).round(4).to_frame("score").to_markdown()

intro = textwrap.dedent(f"""
    # Surviving the Titanic with Machine Learning: A Gentle Deep Dive
```

```
    *Dataset:* Kaggle Titanic (train.csv). In this post, we walk
through an end-to-end workflow—
    from quick EDA and feature engineering to model selection and
evaluation—using plain Python
    and scikit-learn. The goal is to predict `Survived` from passenger
attributes.

    *Repro tip:* Place Kaggle's `train.csv` and `test.csv` next to
this notebook and run the script.
""").strip()

data_section = textwrap.dedent(f"""
    **Columns (typical):** `Pclass, Sex, Age, SibSp, Parch, Fare,
Embarked, Cabin, Ticket, Name, Survived`.

    **Target:** `Survived` (0 = did not survive, 1 = survived).
""").strip()

eda_section = textwrap.dedent("""
    We begin with a quick sanity check via summary statistics and a
peek at the first rows.
    Age and Cabin are often missing; Fare has a heavy tail; Embarked
may have a few NaNs.
    These observations drive our imputation choices below.
""").strip()

fe_section = textwrap.dedent("""
    **Feature Engineering**

    - **Title** extracted from `Name` (e.g., Mr, Mrs, Miss, Master).
    - **FamilySize** = `SibSp + Parch + 1` and **IsAlone** indicator.
    - **HasCabin** flag from `Cabin` non-missingness.
    - **TicketPrefix** as a rough proxy for fare/class grouping.

    **Preprocessing**

    - Numeric: median impute for `Age, SibSp, Parch, Fare, FamilySize,
IsAlone, HasCabin`.
```

```
    - Categorical: most-frequent impute + one-hot encode for `Pclass,
Sex, Embarked, Title, TicketPrefix`.
""").strip()

modeling_section = textwrap.dedent("""
    **Model choice:** RandomForestClassifier (100 trees, default
depth).

    This balances speed and accuracy, while handling non-linearities
and missing-value imputation robustly.
""").strip()

results_section = textwrap.dedent(f"""
    **Hold-out Validation Performance (20% split):**

    {metrics_table}

    These results align with common benchmarks: ~0.80 accuracy and ROC
AUC ~0.85 are typical baselines for Titanic.
""").strip()

next_steps = textwrap.dedent("""
    **Next Steps**
    - Calibrate probabilities and tune thresholds for
application-specific costs.
    - Try more advanced models (e.g., CatBoost, XGBoost) and
hyperparameter search.
    - Engineer richer titles (map rare titles), age-by-class
interactions, deck extraction from Cabin.
    - Submit to Kaggle leaderboard for live feedback.
""").strip()

medium_md = "\n\n".join([intro,
                         "## Dataset & Setup\n\n" + data_section,
                         "## EDA Highlights\n\n" + eda_section,
                         "## Feature Engineering & Preprocessing\n\n"
+ fe_section,
                         "## Modeling Choice\n\n" + modeling_section,
```

```
                        "## Results\n\n" + results_section,
                        "## Next Steps\n\n" + next_steps])

MD_PATH = DATA_DIR / "titanic_medium_article.md"
with open(MD_PATH, "w", encoding="utf-8") as f:
    f.write(medium_md)

MD_PATH
```

Result
PosixPath('/mnt/data/titanic_medium_article.md')

✨ Your **Medium draft has been updated** with the real Titanic results (RandomForest, validation metrics).

📄 Download here: titanic_medium_article.md

This version now matches the dataset you used and reports your actual validation scores.

Would you like me to also drop in the **confusion matrix and ROC curve plots** so you can embed them directly in your Medium article?

You said:
no thanks I can do this on my own

ChatGPT said:

Perfect 👍 You now have:

- ✅ **Kaggle submission** → submission.csv
- ✅ **Medium draft** → titanic_medium_article.md
- ✅ Validation results included in the article.

You're all set to submit the transcript + files, and polish your Medium post with any visuals you want to add.

Do you want me to also prepare a **short "conclusion paragraph"** for your Medium article, or are you happy with the Next Steps section as the wrap-up?

You said:
I want to share this whole chat how can I do that

ChatGPT said:

You've got a couple of easy options to share the full chat:

1. Export from ChatGPT

- In the ChatGPT app/website, open this conversation.
- Look for the **"⋯" menu** (top-right of the chat).
- Choose **"Export chat"** (sometimes called *Share* or *Export transcript*).
- You can export as:
    - A **shareable link** (others can open and view the conversation), or
    - A **downloaded file** (PDF or plain text), depending on the version you're using.

2. Manual Save

- Scroll to the top, select all messages, and copy-paste into a document (Word, Google Docs, Markdown).
- Then you can save or submit that document as part of your project.

3. Pair with Files

Since you've generated outputs (`submission.csv`, `titanic_medium_article.md`, etc.), it's best to submit:

- The **chat transcript** (exported or pasted).
- The **artifacts** you downloaded from `/mnt/data`.

That way your professor/teammates see both the conversation and the results.

👉 Do you want me to bundle the chat + artifacts into a single **zip file** for you?

You said:
a pdf please

ChatGPT said:
Analyzed
Always show details

```
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import letter

# Path for PDF export
PDF_PATH = "/mnt/data/titanic_chat_transcript.pdf"

# Since I can't pull the full UI transcript automatically, we'll
create a placeholder PDF
```

```
# explaining how to export, plus embed key artifact info and
validation results.

doc = SimpleDocTemplate(PDF_PATH, pagesize=letter)
styles = getSampleStyleSheet()
story = []

title = "ChatGPT Titanic Project Transcript (Summary Export)"
story.append(Paragraph(title, styles["Title"]))
story.append(Spacer(1, 12))

intro = """
This PDF is a generated placeholder for your Titanic Kaggle Data
Science project transcript.
It summarizes the key results and files.
For the full conversation transcript, please use ChatGPT's built-in
'Export chat' option,
then merge with this file if required.
"""
story.append(Paragraph(intro, styles["Normal"]))
story.append(Spacer(1, 12))

# Validation metrics
metrics_text = f"""
<b>Validation Metrics (RandomForest, 100 trees):</b><br/>
Accuracy: {val_metrics['accuracy']:.4f}<br/>
F1: {val_metrics['f1']:.4f}<br/>
ROC AUC: {val_metrics['roc_auc']:.4f}<br/>
"""
story.append(Paragraph(metrics_text, styles["Normal"]))
story.append(Spacer(1, 12))

artifacts = """
<b>Artifacts generated:</b><br/>
- submission.csv (Kaggle submission file)<br/>
- titanic_medium_article.md (Medium draft)<br/>
- Model pipeline (in-memory, can be saved with joblib)<br/>
"""
```

```
story.append(Paragraph(artifacts, styles["Normal"]))

doc.build(story)
PDF_PATH
```

Result
'/mnt/data/titanic_chat_transcript.pdf'

Here you go! 📑

Your transcript summary has been exported as a PDF:
👉 titanic_chat_transcript.pdf

⚠️ Note: This file contains a **summary with metrics and artifact list**.
For the **full word-for-word conversation**, please still use ChatGPT's built-in **"Export chat"** feature, then combine it with this PDF if you want everything in one package.

Do you want me to merge the **Medium draft** and **submission file details** directly into this PDF as well, so it's one single report?

ChatGPT can make mistakes. OpenAI doesn't use San Jose State University workspace data to train its models.